User Guide
PUBLIC (PUBLIC)
2020-06-30

SAP Solution Sales Configuration

THE BEST RUN

Content

1 SAP Solution Sales Configuration, On-Premise Edition

Product Information

Product	SAP Solution Sales Configuration, on-premise edition
Release	3.0
Based On	SAP EhP 4 for SAP CRM 7.0 SAP EhP 8 for SAP ECC 6.0
Minimum Supported BI Content Release	BI Content 7.07
Documentation Published	February 2018

Use

SAP Solution Sales Configuration (on-premise edition) is a system that helps customers to configure and sell solutions made of complex product combinations. In addition to selling their own products, many businesses also sell solutions that include products, services, and parts from other manufacturers. For example, many technology companies sell solutions that include combinations of highly complex hardware, software, and services, and each of these can have options or features that must be specified by a customer during the ordering process.

From a modeling perspective, each individual product can be split into many different instances, and there are many possible relationships and dependencies between the product instances. SAP Solution Sales Configuration (on-premise edition) provides a flexible modeling environment, which in turn simplifies the ordering and configuration process for the customer.

SAP Solution Sales Configuration (on-premise edition) includes a configuration engine that provides the ability to perform bottom-up configuration as well as the normal top-down approach. The system offers innovations that make it easier to maintain configuration model data. The system is designed to provide efficient configuration execution performance and advanced integration capabilities.

Integration

SAP Solution Sales Configuration (on-premise edition) is integrated with the following systems:

- SAP Customer Relationship Management (SAP CRM) 7.0 EhP 4
- SAP ERP Central Component 6.0 EhP 8

- Hybris Commerce Suite
  For information about supported versions, see SAP Note 2227752 (Supported Combinations of SAP Commerce Platform and SAP Solution Sales Configuration)

Features

- **Solution Modeling Environment**
  The system provides a modeling environment for creating complex solution models. The solution modeling environment is based on the Eclipse Rich Client Platform (RCP) and can be integrated with SAP CRM and SAP ERP for exchanging configuration master data. Within the modeling environment, you define model elements such as classes and characteristics, and then you define dependencies between your model elements.
  The solution modeling environment can also be integrated with a Source Code Management System to support collaborative modeling and allow several modelers to work on the same model simultaneously.
  The solution modeling environment includes a test user interface and a configuration engine, which allow you to test and optimize your model before transporting it to the target sales system.
  For more information, see Solution Modeling Environment [page 7].
- **Solution Configuration Environment**
  The application provides a configuration engine and a configuration user interface that fully integrate with the ordering process in the SAP CRM and SAP ERP systems. You can enter an advanced mode product (a solution) as an item in a sales document, and then choose to configure it. The system opens the enhanced configuration user interface, where you can enter the solution components and configure their options. When you save the configuration, you are returned to the sales document.
  For more information, see Solution Configuration Environment [page 145].
- **Integration with Non-SAP Sales Systems**
  SAP Solution Sales Configuration (on-premise edition) can also be integrated with non-SAP sales systems. For more information, see Integration with Other Sales Systems [page 191].
- **Solution Configuration Analytics**
  SAP Solution Sales Configuration (on-premise edition) provides a set of DataSources for extracting solution configuration data from the SAP CRM system to a data warehouse.
  For more information, see Solution Configuration Analytics [page 195].

2 Solution Modeling Environment

Use

The solution modeling environment is the part of SAP Solution Sales Configuration that is used to create advanced solution models. It is integrated with SAP Customer Relationship Management (SAP CRM) and SAP ERP Central Component (SAP ECC), where it is used to exchange configuration master data. You define and test your solution model in the solution modeling environment before transporting it to the target system (SAP ECC, SAP CRM, or Hybris), where it is integrated with the sales ordering process and used during solution configuration.

Integration

The solution modeling environment is integrated with the following systems:

- SAP CRM
- SAP ERP

Features

- **Advanced Mode Solution Modeling**
  Models for products, such as a car, are delimited and have a finite number of component parts. Solutions are not delimited; they can have many optional parts with complex dependencies between these parts.
  The solution modeling environment allows you to create advanced solution models. For example, you can define non-part components (instances) without using a bill of material (BOM).
  For information about the elements in a solution model, see Solution Model [page 9]. For information about the steps required to create a solution model, see Solution Modeling [page 73].
- **Data Exchange**
  The solution modeling environment is integrated with SAP CRM and SAP ERP and can be used to exchange configuration master data. For more information, see Data Exchange [page 111].
- **Collaborative Modeling**
  The system supports collaboration between modelers by integrating with standard version control systems. For more information, see Collaboration Between Modelers [page 113].
- **Test User Interface and Test Configuration Engine**
  The system provides a test user interface and a test configuration engine, where you can test and optimize your model before transporting it to the target system. In the test user interface, you can configure your solution to ensure that your model is performing as expected. In addition, you can save test scripts, identify the constraints applied during a configuration session, and test the performance of constraints.
  For more information, see Testing Models Locally [page 93].
- **Knowledge Base Orchestration**

The system supports knowledge base orchestration in the downstream processes in the configuration engine and the configuration user interface. Knowledge base orchestration is a process that enables multiple knowledge bases to handle large configurations. It encapsulates large knowledge base results into smaller, more manageable components and presents multiple independent configuration objects as a single, coordinated configuration session to the end user.

- **Solution Modeling User Interface**
  The solution modeling user interface is based on the Eclipse Rich Client Platform (RCP). It is designed to support different ways of navigation and to perform quickly when modelers are working with large, complex models.
  The solution modeling user interface provides wizards for creating model elements. It also provides a text editor for writing and maintaining element definitions. The text editor has the following features:
  - **Auto-Completion**
    You define model elements with a precise syntax. The text editor provides an auto-completion feature to support modelers when they manually define elements. To use auto-completion, perform the following steps:
    1. Position the cursor at the relevant location and press `CTRL` + `SPACEBAR`.
       The system displays a dialog box listing all of the syntactically correct elements at the current cursor position.
    2. Double-click an element to insert it into the text editor.
  - **Syntax Error Indicator**
    The syntax error icon is displayed on the left side of any lines that contain a syntax error. Furthermore, if syntax errors exist, a red square is displayed in the top right corner of the text editor. You can move the cursor over the red square to see the total number of syntax errors.
  - **Tabbed Display**
    Elements opened in the text editor are displayed on tab pages. You can open several elements for editing and use the tab pages to switch between them.
  - **Templates**
    The solution modeling environment provides templates for common model elements and also allows you to define your own templates. Templates are indicated by a green dot.
    For information about maintaining templates, see Maintaining Modeling Templates [page 112].
- **Data Loader**
  The Data Loader is a component used to download configuration data for your solutions from an SAP ECC or SAP CRM source system to a local database management system such as Microsoft SQL Server. For more information, see Data Loader in the Solution Modeling Environment [page 109].

Constraints

- Certain modeling syntax that works in Variant Configuration (LO-VC) is not supported in the Internet Pricing and Configurator (IPC) configuration engine. Since SAP Solution Sales Configuration is based on the same IPC technology, those restrictions also apply to SAP Solution Sales Configuration. For more information about the restrictions, refer to SAP Note 1819856 (Additions to IPC VC deltalist).
- Solution modeling can be very complex as the requirements and dependencies to be considered when you assemble a solution can be both numerous and complex. While the configuration engine in SAP Solution Sales Configuration is built specifically to handle these processing intensive requirements, the critical factor in ensuring efficient processing is the definition of the model. Great care must be taken in developing solution models. It is a similar process to any other application development and requires training, abstract thought, sophistication, and elegance of design.

2.1 Introduction to Solution Models

Definition

A solution model is a hierarchical decomposition of a solution. It defines the products (configurable materials and the services) that can be contained within the solution. It also defines the relationships between the various elements of the solution, including any dependencies (constraints and rules).

Structure

A solution model contains the following parts:

- **Knowledge Bases (KBs)**
  For more information, see Knowledge Base [page 10].
  For information about defining knowledge bases, see Defining Knowledge Bases [page 10].
- **Classes**
  For more information, see Class [page 15].
  For information about defining classes, see Defining Classes [page 15].
- **Materials**
  For more information, see Material [page 20].
  For information about defining materials, see Defining Materials [page 20].
- **Characteristics (cstics)**
  For more information, see Characteristic [page 24].
  For information about defining characteristics, see Defining Characteristics [page 25].
- **Variant Tables**
  For more information, see Variant Table [page 38].
  For information about defining variant tables, see Defining Variant Tables [page 44].
- **Dependency Nets**
  For more information, see Dependency Net [page 55].
  For information about defining dependency nets, see Defining Solution Dependencies [page 47].
- **User-Defined Functions**
  For more information, see User-Defined Function [page 56].
- **Interface Designs**
  For more information, see Interface Design [page 64].
  For information about creating interface designs, see Example: Defining an Interface Design [page 64].
- **Bills of Material (BOMs)**
  For more information, see Bill of Material [page 65].
  For information about defining BOMs, see Defining Dynamic BOMs [page 66].

2.1.1 Knowledge Base

Definition

A knowledge base contains the master data for a solution model. The knowledge base is exported from the solution modeling environment, initially to the test user interface and the test configuration engine to be tested, and then to the target sales system where it is used by the configuration engine.

Structure

In the solution modeling environment, a knowledge base is defined with identifier, version, tasks, and profiles.

More Information

For information about defining knowledge bases, see Defining Knowledge Bases [page 10].

2.1.1.1 Defining Knowledge Bases

Use

You use this procedure to create a task, which is then used to define a knowledge base. You can define the task in the same file as the knowledge base, or in a separate file.

Procedure

Creating a Knowledge Base

1. Choose *File* › *New* › *Empty Model File*.
   The system displays a dialog prompting you to choose a folder and enter a file name for the new knowledge base.
2. Enter the required data.
3. Choose *Finish*.
   The system displays an empty file.
4. Press CTRL + SPACEBAR.
5. Double-click *knowledgeBase*.
6. Enter the required data and save the knowledge base.

2.1.1.2 Wissensbasis-Laufzeitversion

Eine Wissensbasis-Laufzeitversion wird aus einer Wissensbasisdefinition angelegt, indem die Funktion *Wissensbasis exportieren* in der Lösungsmodellierungsumgebung aufgerufen wird.

Sie können die Funktion *Wissensbasis exportieren* aufrufen:

- aus dem Menü *Datei*, indem Sie *Datei* > *Exportieren* > *SAP Solution Configuration* > *Wissensbasis exportieren* wählen.
- aus dem Modellexplorer, indem Sie mit der rechten Maustaste auf eine Wissensbasisdefinition klicken und *Wissensbasis exportieren* wählen.

Wenn Sie eine Wissensbasis exportieren, können Sie Variantentabelleninhalt in die WB-Laufzeitversion einschließen, auch wenn das Schlüsselwort `externalTable` in der Variantentabellendefinition angegeben wurde. Dies ist hilfreich, um die Wissensbasis lokal zu testen. Wenn externer Inhalt verfügbar ist und Inhalt auch in die Wissensbasis eingeschlossen wurde, wird der **externe** Inhalt zur Laufzeit verwendet.

Um lokalen Inhalt in die WB-Laufzeitversion einzuschließen (entweder aus „Zeilen"-Werten oder „Datei"-Inhalt), wählen Sie die Option *Inhalt der lokalen Variantentabelle in Export einschließen* im Exportdialogfenster der WB oder *Lokale VT* im Dialogfenster zum Exportieren mehrerer Wissensbasen.

2.1.1.3 Example: Defining a Knowledge Base Definition

i Note

This description uses the following conventions to illustrate the syntax requirements:

Required keyword

Optional keyword

`User-specified value`

`Mutually exclusive keyword`

A knowledge base definition indicates the materials or classes for which the knowledge base will be used. It also lists the dependencies to be included in the knowledge base runtime version. To define a knowledge base definition, use the following syntax:

knowledgeBase **`identifier`** {

version **`kb-version`**

*validFrom***`date`**

logsys **`logical-system`**

tasks **`task-id`**

profiles

name **`profile-name`** *class class-id* || *material material-id*

}

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`knowledgeBase`	Denotes the start of a knowledge base definition encompassed within **{...}**	-	Required
`identifier`	Knowledge base ID used in the model. Max. length 30 bytes It appears in the `KBOBJNAME` field in the `comm_cfgkb` table.	-	Required
`version`	Keyword	-	Required
`kb-version`	Version is a text field; common format is “0.0.0.0” but any alternative format is acceptable.	-	Required
`logsys`	Keyword	-	Optional
`logical-system`	Logical system name where the materials/classes named in profiles (below) originated. The logical system value is derived from the back end. All materials defined for profiles of this knowledge base are checked. If multiple entries exist, a list is presented for selection by the user.	-	Optional
`validFrom date`	Date in the format “YYYY-MM-DD”from which this knowledge base will be effective from. If `validFrom` is not specified, the current date is used.	Required with `validFrom` keyword	
`tasks`	Keyword	-	Optional
`task-id`	ID of task containing the dependencies to be included in this knowledge base	Required with `tasks` keyword	

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`profiles`	Denotes the start of the list of profiles. This list can contain multiple profiles. Each name-class/material pair is a distinct profile definition.	-	Required
`name`	Keyword	-	Required
`'profile-name'`	The profile name is a key value for the knowledge base tables.	-	Required
`class`	Keyword	At least one class or material must be specified	Required
`class-id`	A class-id defined in this model	-	Required with class keyword
`material`	Keyword	At least one class or material must be specified	Required
`material-id`	A material-id defined in this model	-	Required with material keyword

Example

```
knowledgeBase PRODUCT_X_KB {
   version "0.0.0.6"
   validFrom "2020-01-01"    logsys "LOCAL"
   tasks
      PRODUCT_X_TSK
   profiles
   name 'PRODUCT_X_PRF_2' class PRODUCT_X
}
```

2.1.1.4 Definition of External Texts

Use

After you have exported a knowledge base to the SAP back-end system, you can create or change the texts that have been defined for each characteristic, characteristic value, class, or material. Editing the texts directly in the back end means that you do not have to export your knowledge base again if you want to make changes.

i Note

A particularity of maintaining material texts externally is that if a material is also defined in the product master in the SAP CRM back end, the text from the product master is taken automatically. This avoids the need for double maintenance. If the material is not defined in the back end, you can maintain the texts manually in the same way as all of the other objects.

Integration

You can maintain external texts manually in Customizing or you can export them from the Solution Modeling Environment as follows:

1. Choose *File* > *Export* > *Export Knowledge Base Localization*.
2. Choose a connection.
3. Select your knowledge base and, if required, enter a date from which the texts are valid.

Activities

You maintain the texts in Customizing for CRM under *Basic Functions* > *Solution Sales Configuration* > *Maintain External Texts*.

You maintain the texts in Customizing for ECC under *Logistics - General* > *Solution Sales Configuration* > *Maintain External Texts*.

Trigger the customizing download from SME to download external texts into the local database.

Specify language code while opening a knowledge base in SME. On selecting *Show Localized Names* you can view the external text in SME.

i Note

If you want to load external texts while testing configuration in SME, specify the engine flag *isExternalTextEnabled* as *True*. Refer SAP Note 2341032 and 2291607.

Example

In your knowledge base, you have a characteristic with the names COLOR and FARBE for English and German respectively. After you have exported your knowledge base to the SAP CRM back end, you can localize the characteristic for the United Kingdom by changing COLOR to COLOUR, and you can also add a new translation COULEUR for French.

More Information

For more information, see the Customizing documentation in the system.

2.1.2 Klassen

Klassen werden zur Darstellung der unterschiedlichen konfigurierbaren Teile eines Lösungsmodells verwendet. Klassen können in einer Hierarchie definiert werden, wobei Unterklassen die Merkmale ihrer übergeordneten Klasse übernehmen. Sie können beispielsweise eine Klasse mit der Bezeichnung "Computer" und den beiden Unterklassen "Laptop" und "Desktop" definieren.

Struktur

Eine Klasse hat eine ID, einen Namen und ein oder mehrere Merkmal(e).

Weitere Informationen

Weitere Informationen zu Merkmalen erhalten Sie unter Merkmale [Seite 24].

Weitere Informationen zum Definieren von Klassen finden Sie unter Definieren von Klassen [Seite 15].

2.1.2.1 Definieren von Klassen

Verwendung

Mit diesem Verfahren definieren Sie neue Klassen in der Eclipse-basierten Lösungsmodellierungsumgebung.

Vorgehensweise

Anlegen einer neuen Klasse über das Hauptmenü

1. Wählen Sie *Datei* > *Neu* > *Klasse*.
2. Geben Sie die erforderlichen Daten ein.

3. Wählen Sie *Fertigstellen*.
   Die neue Klasse wird im Texteditor geöffnet.

Anlegen einer neuen Klasse über das Kontextmenü

1. Klicken Sie im *Projektexplorer* mit der rechten Maustaste auf *Klassen*.
   Das Kontextmenü wird angezeigt.
2. Wählen Sie *Neu* > *Andere...*
3. Wählen Sie *Klasse* und dann *Weiter*.

   > **i Note**
   >
   > Wenn Sie *Klasse* einzugeben beginnen, wird der Inhalt des Explorers gefiltert, sodass nur die passenden Einträge angezeigt werden.

4. Geben Sie die erforderlichen Daten ein.
5. Wählen Sie *Fertigstellen.*
   Die neue Klasse wird im Texteditor geöffnet.

Hinzufügen von Merkmalen zu einer Klasse über den Texteditor

1. Öffnen Sie die Klasse im Texteditor.
2. Platzieren Sie den Cursor an der gewünschten Stelle.

   > **i Note**
   >
   > Merkmale werden in einer kommagetrennten Liste unter *Merkmale* eingegeben.

3. Drücken Sie STRG + LEERTASTE.
   Es wird eine Liste aller verfügbaren Merkmale angezeigt.
4. Doppelklicken Sie auf das Merkmal, das Sie der Klasse hinzufügen möchten.

Hinzufügen weiterer Informationen für ein Merkmal

1. Platzieren Sie im Texteditor den Cursor hinter dem Merkmal.
2. Geben Sie ein Leerzeichen ein und drücken STRG + LEERTASTE.
   Es werden folgende Optionen angezeigt:
   - **Standard**
     Mit dieser Option geben Sie einen Standardwert für das Merkmal an.
   - **Unsichtbar**
     Mit dieser Option blenden Sie das Merkmal in der Konfigurationsbenutzungsoberfläche aus.
   - **Erforderlich**
     Mit dieser Option machen Sie das Merkmal in der Konfigurationsbenutzungsoberfläche obligatorisch.
   - **Wert**
     Mit dieser Option geben Sie den Wert des Merkmals an.
3. Doppelklicken Sie auf die Option, die Sie dem Merkmal hinzufügen möchten.
4. Sie müssen einen Wert für die Optionen *Standard* und *Wert* festlegen.

Beispiel

class SME_EXAMPLE_MEMORY {

name "Memory

characteristics

SME_INSTANCE_NF **invisible**,

SME_MEMORY_SPEED_S **required**,

SME_PART_NUMBER_S **required**

}

2.1.2.2 Defining a Class

> **i Note**
>
> This description uses the following conventions to illustrate the syntax requirements:
>
> *Required keyword*
>
> **Optional keyword**
>
> **`User-specified value`**
>
> `Mutually exclusive keyword`

To define a class, use the following syntax:

class **`identifier`** **extends** **`super-class-id, super-class-id2,...`** {

name **`"30 bytes descriptive text"`** || names EN **`"descriptive English text"`**

, FR **`"descriptive French text"`**

longName **`"unlimited descriptive text"`** || longNames EN **`"unlimited text in multiple languages"`**

urls { **`"url"`** *label* **`"30 bytes descriptive text", "url"`** *label* **`"30 bytes descriptive text" ...`** **}**

> **i Note**
>
> The `urls` keyword can be used at group, cstic, cstic value, class, and material level. As a general rule of thumb, all elements that could be UI relevant have this attribute. It is also possible to define more than one URL by using curly brackets.

characteristics **`characteristic-id`**

--remaining keywords are specified for each characteristic on this class definition--

required

noinput

urls { **`"url"`** *label* **`"30 bytes descriptive text", "url"`** *label* **`"30 bytes descriptive text" ...`** **}**

values (**`value1, value2,...`** **)**

defaultValues(**`value1,value2,...`** **)**

```
assignedValues( value1,value2,... )
}
```

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`class`	Denotes the start of a class definition encompassed within {...}	-	Required
`identifier`	This is the class ID used in the model to reference this class. It must be uppercase. Max. length 18 bytes	-	Required
`extends`	Keyword indicating inheritance from a super-class	-	Optional
`super-class-id`	Identifier of the class extended by this definition	Required if extends keyword is used	Optional
`super-class-id2`	Identifier(s) of additional classes extended by this definition. Multi-inheritance is supported.	-	Optional
`{`	Starts the set of keywords that define this object	-	Required
`}`	Ends class definition	-	Required
`name` or `names`	The descriptive, language-dependent text used as a label for this object. If not used, the identifier is used as the name. See the context-sensitive help for the full list of available two-character language identifiers.	-	Optional
`longName` or `longNames`	Unlimited length text available from the "more information" link on the configuration UI. It is possible to embed hyperlinks in this text.	-	Optional

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`characteristics`	Keyword indicating the start of the list of characteristics for this class	-	Optional
The following keywords are specified for each characteristic			
`required`	Indicates that the configuration is considered incomplete unless a value is specified for this characteristic	-	Optional
`noinput`	Blocks the field from user input. Values may still be assigned/changed by means of constraints or rules.	-	Optional
`defaultValues()`	Sets the default value(s) of this characteristic for this class. More than one value may be specified only if the characteristic is defined as multiValue.		Optional
`assignedValues()`	Assigns a value (or set of values) to this characteristic for this class. More than one value may be specified only if the characteristic is defined as multiValue. This is a fixed assignment. The value(s) cannot be changed by the user, constraints, or rules.	-	Optional

Example

```
class PRODUCT_Y extends SSC_SD_HIER {
name "Product Y class"
characteristics
SALES_ITEM_NAME      required noinput,
NUM_SUBSCRIBERS      required defaultValues (5000),
TRAFFIC_PER_SUBSCRIBER  required assignedValues (25),
TOTAL_TRAFFIC           invisible noinput
}
```

2.1.3 Material

Ein Material ist eine grundlegende Komponente eines Lösungsmodells.

In der Eclipse-basierten Lösungsmodellierungsumgebung definieren Modeler ein oder mehrere Material(ien), die dann in Constraints und Regeln referenziert werden können.

Materialien sind, zusammen mit Produkten und Klassen, über ein Profil in der Wissensbasisdefinition mit einer Wissensbasis verknüpft.

Struktur

Materialien haben eine Klasse und eine ID.

Weitere Informationen

Weitere Informationen zum Definieren von Materialien finden Sie unter Definieren von Materialien [Seite 20].

2.1.3.1 Definieren von Materialien

Kontext

Mit diesem Verfahren definieren Sie neue Materialien in der Eclipse-basierten Lösungsmodellierungsumgebung.

Vorgehensweise

1. Wählen Sie *Datei* > *Neu* > *Leere Modelldatei*

   Es wird ein Dialogfenster angezeigt, in dem Sie dazu aufgefordert werden, einen Ordner auszuwählen und einen Dateinamen für das neue Material einzugeben.
2. Geben Sie die erforderlichen Daten ein.
3. Wählen Sie *Fertigstellen*.

   Im Texteditor wird eine leere Datei angezeigt.

4. Drücken Sie STRG + LEERTASTE, und doppelklicken Sie auf *material*.

   > i Note
   >
   > Sie können auch **`material`** eingeben.

5. Geben Sie die erforderlichen Daten ein, und sichern Sie das neue Material.

Beispiel

material SME_LAPTOP {

name 'Laptop'

classes

SME_LAPTOP

boms

SME_LAPTOP

}

2.1.3.2 Beispiel: Definieren von Material

> i Note
>
> Diese Beschreibung verwendet die folgenden Konventionen zum Darstellen der Syntaxanforderungen:
>
> *Erforderliches Schlüsselwort*
>
> **Optionales Schlüsselwort**
>
> **`Benutzerdefinierter Wert`**
>
> `Einander ausschließende Schlüsselwörter`

Verwenden Sie zum Definieren von Material die folgende Syntax:

*material***`identifier`****externalID** **`'external_identifier'`***{*

name**`"beschreibender Text 30 Byte"`** || **names** DE **`"beschreibender deutscher Text"`**

, FR **`"beschreibender`**

`französischer Text"`

longName**`"unbeschränkter beschreibender Text"`** || **longNames** EN **`"unbeschränkter Text in mehreren Sprachen"`**

*urls {***`"URL"`***label***`"beschreibender Text 30 Byte"`**,**`"URL"`***label***`"beschreibender Text 30 Byte"...}`**

*classes***`class-id`**

```
bomsbom-id
}
```

Schlüsselwort/Benutzer-wert	Bedeutung	Beziehung zu anderen Schlüsselwörtern	Erforderlich/Optional
`class`	Kennzeichnet den Anfang einer Klassendefinition		Erforderlich
`identifier`	Material-ID, die im Modell zur Referenzierung dieses Materials verwendet wird. Max. Länge 18 Byte		Erforderlich
`external_ID`	Schlüsselwort, das angibt, dass CRM oder ECC eine alternative ID zur Referenzierung dieses Materials verwendet		Optional
`external_idenitfier`	ID, die von CRM oder ECC zur Referenzierung dieses Materials verwendet wird	Erforderlich, wenn das Schlüsselwort `external_ID` verwendet wird	Optional
`{`	Steht am Anfang der Reihe von Schlüsselwörtern, die dieses Objekt definieren		Erforderlich
`}`	Beendet die Klassendefinition		Erforderlich
`name` oder `names`	Beschreibender, sprachabhängiger Text, der als Bezeichnung für dieses Objekt verwendet wird. Wird dieser nicht verwendet, wird die ID als Name verwendet. Die vollständige Liste der verfügbaren zweistelligen Sprachen-IDs finden Sie in der kontextsensitiven Hilfe.		Optional

Schlüsselwort/Benutzerwert	Bedeutung	Beziehung zu anderen Schlüsselwörtern	Erforderlich/Optional
`longName` oder `longNames`	Text mit unbeschränkter Länge verfügbar über den Link „Weitere Informationen“ in der Konfigurationsbenutzungsoberfläche. In diesen Text können Hyperlinks eingebettet werden.		Optional
`classes`	Schlüsselwort, das den Anfang der Liste von Klassen angibt, die diesem Material zugeordnet sind		Optional
`class-id`	Eine oder mehrere diesem Material zugeordnete Klassen-IDs. Mehrere Klassen durch Komma trennen.	Erforderlich, wenn das Schlüsselwort `classes` verwendet wird	Optional
`boms`	Schlüsselwort, das die mit diesem Baugruppenmaterial verknüpfte Stückliste angibt. (Hinweis: Stücklisten werden nicht empfohlen; als Best Practice wird die Verwendung von ADT zur Darstellung von „Teil-von“-Beziehungen empfohlen.)		Optional
`bom`	Stückliste, die zur Darstellung der Komponenten dieses Baugruppenmaterials definiert wurde	Erforderlich, wenn das Schlüsselwort `boms` verwendet wird	Optional

Beispiel

```
material CABLE_SET {
 name "Cable set"
 classes
  CABLE_SET
}
```

2.1.4 Characteristic

Definition

Characteristics are the lowest level of detail in a solution model. They are used to define the distinguishing properties or attributes of a class. For example, a computer monitor can have a characteristic called size.

Structure

A characteristic has an identifier, a name, and a type. Depending on the type selected, you may need to specify additional information, as shown in the following table:

Characteristic Types

Characteristic Type	Information to Be Defined
Text	Length
Numeric	Length Decimal Places
Date	
ADT	Class

Reference Characteristics

Characteristics of type *Text*, *Numeric*, and *Date* can be made reference characteristics by using the `reference` keyword. Reference characteristics can be used to do the following:

- Pass a value to the configuration as a context
- Pass a value from the configuration to the sales document
  In this case, the user is usually required to enter the value of the reference characteristic during the configuration session (for example, size, weight, or start date).

Abstract Data Types

An abstract data type (ADT) is a special type of characteristic that is used to define the relationship between two classes. For example, a computer monitor can have ADTs called **has part** and **is part of**.

SAP Solution Sales Configuration provides the following predefined ADTs:

- `SALES HARD`
- `SALES SOFT`

These ADTs are used to define parent-child relationships as either a **hard tie** or a **soft tie**. If two classes have a hard tie, they cannot be split into separate sales documents. Only classes with soft ties can be split into different sales documents.

You maintain the names of the hard and soft tie sales ADTs in Customizing for *Customer Relationship Management* under *Basic Functions* > *Solution Sales Configuration* > *Maintain Item Relationship Types*.

More Information

For information about defining characteristics, see Defining Characteristics [page 25].

2.1.4.1 Definieren von Merkmalen

Verwendung

Mit diesem Verfahren definieren Sie neue Merkmale in der Eclipse-basierten Lösungsmodellierungsumgebung.

Vorgehensweise

Definieren von Merkmalen im Hauptmenü

1. Wählen Sie im Hauptmenü *Datei* > *Neu* > *Merkmal*.
2. Geben Sie die erforderlichen Daten ein.

   > **i Note**
   >
   > Ein *Identifikator* muss aus alphanumerischen Zeichen in Großschrift und Unterstrichen bestehen; am Anfang darf kein numerisches Zeichen stehen. Der *Name* entspricht standardmäßig demselben Wert wie der *Identifikator*; er kann bei Bedarf geändert werden.

3. Wählen Sie *Fertigstellen*.
   Das neue Merkmal wird im Texteditor geöffnet.

Definieren von Merkmalen im Kontextmenü

1. Klicken Sie im *Projektexplorer* mit der rechten Maustaste auf *cstics*, um das Kontextmenü anzuzeigen.
2. Wählen Sie *Neu* > *Andere...*
3. Wählen Sie *Merkmal.*

   > **i Note**
   >
   > Wenn Sie *Merkmal* einzugeben beginnen, wird der Inhalt des Explorers gefiltert, sodass nur die passenden Einträge angezeigt werden.

4. Wählen Sie *Weiter.*
5. Geben Sie die erforderlichen Daten ein.
6. Wählen Sie *Fertigstellen.*
   Das neue Merkmal wird im Texteditor geöffnet.

2.1.4.2 Defining a Text Characteristic

> Note
>
> This description uses the following conventions to illustrate the syntax requirements:
>
> *Required keyword*
>
> **Optional keyword**
>
> **`User-specified value`**
>
> `Mutually exclusive keyword`

To define a text characteristic, use the following syntax:

characteristic **`identifier`** *{*

name **`"30 bytes descriptive text"`** || **names EN** **`"descriptive English text"`**

, **FR** **`"descriptive French text"`**

longName **`"unlimited descriptive text"`** || **longNames EN** **`"unlimited text in multiple languages"`**

status released || **prepared** || **blocked**

textLength **`1-132`**

restrictable `(x-additionalValues)`

caseSensitive

multiValue

additionalValues `(x-restrictable)`

`values` "a" **name** **`"Aye"`** || **names EN** **`"descriptive English text"`**

, **FR** **`"descriptive French text"`**

`, "b"` **name** **`"Bee"`** || **names EN**

}

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`characteristic`	Denotes the start of a characteristic definition encompassed within {...}	-	Required
`identifier`	This is the characteristic ID used in the model to reference this characteristic. It must be uppercase. Max. length 40 bytes	-	Required

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`name` or `names`	The descriptive, language-dependent text used as a label for this object. If not used, the identifier is used as the name. See the context-sensitive help for the full list of available two-character language identifiers.	-	Optional
`longName` or `longNames`	Unlimited length text available from the “more information” link on the configuration UI. It is possible to embed hyperlinks in this text.	If the name keyword is not used, the longName is **not** used as the name.	Optional
`status`	If used, one of three additional keywords must be specified - required, prepared, blocked. This keyword is ignored by the SME. It is available for compatibility with master data definitions imported from ECC.	-	Optional
`textLength`	Indicates that this is a text characteristic and specifies the maximum length of the value. Maximum `textLength` is 132.	-	Required
`restrictable`	Indicates that the domain of values can be restricted. If the `values` clause is used, the domain is restricted to those values, and can be further restricted in constraints. If no `values` clause is used, the initial static domain is the “unrestricted domain”. Any value is valid, but can still be restricted by constraints.	This clause and the `additionalValues` clause are mutually exclusive.	Optional

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`caseSensitive`	Indicates that values can contain uppercase and lowercase values	-	Optional
`multiValue`	Indicates that more than one value can be assigned to this characteristic	-	Optional
`additionalValues`	Allows user entry of values. Can be used with the values clause. The values specified in the `values` clause are presented for selection, but the user can also enter a different value if the `additionalValues` clause is specified.	This clause and the `restrictable` clause are mutually exclusive.	Optional
`values`	Lists the permitted values. No values indicates an unrestricted domain. Any type-consistent value is allowed. Values can be specified with or without names and names can be specified in multiple languages.	-	Optional

Example

```
characteristic CABINET_TYPE {
      name "Cabinet Type"
      textLength 1
      caseSensitive
      restrictable
      values
         'a' name 'little aye',
         'A' name 'Aye',
         'B' name 'Bee',
         'C' name 'See',
         'D' name 'Dee',
         'd' name 'little dee'
}
```

2.1.4.3 Defining a Numeric Characteristic

> **Note**
>
> This description uses the following conventions to illustrate the syntax requirements:
>
> *Required keyword*
>
> **Optional keyword**
>
> **`User-specified value`**
>
> `Mutually exclusive keyword`

To define a numeric characteristic, use the following syntax:

characteristic **`identifier`** {

name **`"30 bytes descriptive text"`** || **names EN** **`"descriptive English text"`**

, **FR** **`"descriptive French text"`**

longName **`"unlimited descriptive text"`** || **longNames EN** **`"unlimited text in multiple languages"`**

numericLength **`1-15`**

decimalPlaces **`0-(numericLength-1)`**

restrictable `(x-additionalValues)`

multiValue

negativeValues

additionalValues `(x-restrictable)`

specialFunction aggregating

intervals < n1 || <= n1 || > n2 || >= n2 || n1 - n2 || (where n1, n2 are numeric values)

values **`n1, n2`**

}

Keyword/User Value	Meaning	Required/Optional
`characteristic`	Denotes the start of a characteristic definition encompassed within {...}	Required
`identifier`	This is the characteristic ID used in the model to reference this characteristic. It must be uppercase. Max. length 30 bytes	Required

Keyword/User Value	Meaning	Required/Optional
`name` or `names`	The descriptive, language-dependent text used as a label for this object. If not used, the identifier is used as the name. See the context-sensitive help for the full list of available two-character language identifiers.	Optional
`longName` or `longNames`	Unlimited length text available from the "more information" link on the configuration UI. It is possible to embed hyperlinks in this text.	Optional
`numericLength`	Indicates that this is a numeric characteristic and specifies the maximum length of the value. Maximum `numericLength` is 15.	Required
`decimalPlaces`	Indicates the number of decimal places of this numeric value. Maximum `decimalPlaces` is one less than the `numericLength`.	Optional
`restrictable`	Indicates that the domain of values can be restricted. If the `values` clause is used, the domain is restricted to those values, and can be further restricted in constraints. If no `values` clause is used, the initial static domain is the "unrestricted domain". Any value is valid, but can still be restricted by constraints.	Optional
`multiValue`	Indicates that more than one value can be assigned to this characteristic	Optional
`negativeValues`	Allows a numeric value to be a negative value	
`additionalValues`	Allows user entry of values. Can be used with the values clause. The values specified in the `values` clause are presented for selection, but the user can also enter a different value if the `additionalValues` clause is specified.	Optional

Keyword/User Value	Meaning	Required/Optional
`specialFunction aggregating`	Designates this characteristic as usable in a rule with the `then_increment` statement	Optional
`specialFunction balanced`	Do not use - not supported	Prohibited
`values`	Lists the permitted values. No values indicates an unrestricted domain. Any type-consistent value is allowed	Optional
`specialFunction default`	Designates this value as the default va-lue	Optional

Example

```
characteristic BLOCKING {
      name "Blocking"
      numericLength 2 decimalPlaces 1
      restrictable
      values 0.1, 0.5, 1, 2, 5
}
```

2.1.4.4 Defining a Date Characteristic

i Note

This description uses the following conventions to illustrate the syntax requirements:

Required keyword

Optional keyword

`User-specified value`

`Mutually exclusive keyword`

To define a date characteristic, use the following syntax:

characteristic **`identifier`** *{*

name **`"30 bytes descriptive text"`** || names EN **`"descriptive English text"`**

, FR **`"descriptive French text"`**

longName **`"unlimited descriptive text"`** || longNames EN **`"unlimited text in multiple languages"`** *date*

restrictable (`x-additionalValues`)

multiValue

additionalValues (`x-restrictable`)

values **`yyyy-mm-dd`**

}

Keyword/User Value	Meaning	Required/Optional
`characteristic`	Denotes the start of a characteristic definition encompassed within {...}	Required
`identifier`	This is the characteristic ID used in the model to reference this characteristic. It must be uppercase. Max. length 30 bytes	Required
`{`	Starts the set of keywords that define this object	Required
`}`	Ends the object (characteristic) definition	Required
`name` or `names`	The descriptive, language-dependent text used as a label for this object. If not used, the identifier is used as the name. See the context-sensitive help for the full list of available two-character language identifiers.	Optional
`longName` or `longNames`	Unlimited length text available from the “more information” link on the configuration UI. It is possible to embed hyperlinks in this text.	Optional
`date`	Indicates that this is a date characteristic	Required

Keyword/User Value	Meaning	Required/Optional
`restrictable`	Indicates that the domain of values can be restricted. If the `values` clause is used, the domain is restricted to those values, and can be further restricted in constraints. If no `values` clause is used, the initial static domain is the "unrestricted domain". Any value is valid, but can still be restricted by constraints.	Optional
`multiValue`	Indicates that more than one value can be assigned to this characteristic	Optional
`additionalValues`	Allows user entry of values. Can be used with the values clause. The values specified in the `values` clause are presented for selection, but the user can also enter a different value if the `additionalValues` clause is specified.	Optional
`values`	Lists the permitted values. No values indicates an unrestricted domain. i Note The date must be specified in "YYYY-MM-DD"format only. Any type-consistent value is allowed.	Optional

Example

```
characteristic BIRTHDAY {
      name "Date of Birth"
      date
      values
       1992-12-25
}
```

2.1.4.5 Defining an Abstract Data Type Characteristic

> **i Note**
>
> This description uses the following conventions to illustrate the syntax requirements:
>
> *Required keyword*
>
> **Optional keyword**
>
> **`User-specified value`**
>
> `Mutually exclusive keyword`

To define an abstract data type (ADT) characteristic, use the following syntax:

characteristic **`identifier`** *{*

name **`"30 bytes descriptive text"`** || **names EN** **`"descriptive English text"`**

, **FR** **`"descriptive French text"`**

longName **`"unlimited descriptive text"`** || **longNames EN** **`"unlimited text in multiple languages"`**

classType *class-identifier*

multiValue

}

Keyword/User Value	Meaning	Required/Optional
`characteristic`	Denotes the start of a characteristic definition encompassed within {...}	Required
`identifier`	This is the characteristic ID used in the model to reference this characteristic. It must be uppercase. Max. length 30 bytes	Required
`name` or `names`	The descriptive, language-dependent text used as a label for this object. If not used, the identifier is used as the name. See the context-sensitive help for the full list of available two-character language identifiers.	Optional

Keyword/User Value	Meaning	Required/Optional
`longName` or `longNames`	Unlimited length text available from the "more information" link on the configuration UI. It is possible to embed hyperlinks in this text.	Optional
`classType`	Indicates that this is an abstract data type characteristic	Required
`class-identifier`	Identifier of the class referenced by this characteristic	Required
`multiValue`	Indicates that more than one value can be assigned to this characteristic	Optional

Example

```
characteristic CONTAINS_THESE_ITEMS {
 name "Contains these items"
 classType ITEM
 multiValue
}
```

2.1.4.6 Defining a Reference Characteristic

i Note

This description uses the following conventions to illustrate the syntax requirements:

Required keyword

Optional keyword

`User-specified value`

`Mutually exclusive keyword`

A reference characteristic refers to the value of a field in the sales document. To define a reference characteristic, use the following syntax:

characteristic **`identifier`** *{*

name **`"30 bytes descriptive text"`** || **names EN** **`"descriptive English text"`**

, **FR** **`"descriptive French text"`** longName **`"unlimited descriptive text"`** || longNames EN **`"unlimited text in multiple languages"`**

textLength **`1-30`** || *numericLength* **`1-15`** *decimalPlaces* **`0-(numericLength-1)`** || *date*

reference table **`table-name`**

field **`field-name`**

}

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`characteristic`	Denotes the start of a characteristic definition encompassed within **{...}**	-	Required
`identifier`	This is the characteristic ID used in the model to reference this characteristic. It must be uppercase. Max. length 30 bytes	-	Required
`name` or `names`	The descriptive, language-dependent text used as a label for this object. If not used, the identifier is used as the name. See the context-sensitive help for the full list of available two-character language identifiers.	-	Optional
`longName` or `longNames`	Unlimited length text available from the "more information" link on the configuration UI. It is possible to embed hyperlinks in this text.	-	Optional
`textLength`	Indicates that this is a text characteristic and specifies the maximum length of the value. Maximum `textLength` is 30.	One of the `numericLength` / `decimalPlaces`, `textLength`, or `date` keywords is required	Required

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`numericLength`	Indicates that this is a numeric characteristic and specifies the maximum length of the value. Maximum `numericLength` is 15.	One of the `numericLength` / `decimalPlaces`, `textLength`, or `date` keywords is required	Required
`decimalPlaces`	Indicates the number of decimal places in this numeric value. Maximum `decimalPlaces` is one less than the `numericLength`.	Allowed only with `numericLength` keyword	Optional
`date`	Indicates that this is a date characteristic	One of the `numericLength` / `decimalPlaces`, `textLength`, or `date` keywords is required	Required
`reference table`	Indicates the table containing the field to which this characteristic refers	-	Required
`field`	Indicates the field to which this characteristic refers	-	Required

Example

```
characteristic ITEM_QUANTITY {
names
      EN 'Component quantity',
      DE 'Komponentenmenge'
numericLength 13 decimalPlaces 3
negativeValues
reference table 'STPO' field 'MENGE'
}
```

2.1.5 Variantentabellen

Definition

Variantentabellen werden zum Speichern von Kombinationen aus Werten für Merkmale verwendet. Sie können zum Ableiten von Werten als Teil von Constraints verwendet werden.

Struktur

Sie definieren eine Variantentabelle für ein oder mehrere Merkmal(e), wobei jedes Merkmal eine Spalte in der Variantentabelle enthält. Sie geben für jede mögliche Kombination von Merkmalswerten eine Zeile in die Variantentabelle ein. Durch Eingabe eines der Merkmale wird die Selektion der anderen Merkmale eingeschränkt.

Sie definieren eine Variantentabelle in der Lösungsmodellierungsumgebung, indem Sie `row`-, `file`- oder `externalTable`-Schlüsselwörter verwenden.

> **i Note**
>
> Das `externalTable`-Schlüsselwort verweist auf eine Tabelle aus dem ABAP-Backend-System. Falls diese Tabelle vorhanden ist, wird deren Inhalt zur Laufzeit anstelle des Inhalts, der Bestandteil der Laufzeitversion der Wissendatenbank ist, geladen.
>
> Wenn Sie eine Tabelle aus dem ABAP-Backend-System eingeben, müssen Sie Von- und Bis-Daten sowie den Mandanten (ABAP-Tabellen sollten auf einer mandantenspezifischen Basis angelegt werden). Die gültigen Von- und Bis-Datumsspalten müssen als `SSC_FROM_DATE` und `SSC_TO_DATE` benannt werden und vom Datentyp `DATS` sein.

Beispiel:

Im folgenden Beispiel kann die Speicherbeschreibung verwendet werden, um die Speichergröße und -geschwindigkeit abzuleiten:

Speicherbeschreibung	Speichergröße	Speichergeschwindigkeit
`1_1333`	1	1333
`2_1333`	2	1333
`2_1600`	2	1600
`2_2000`	2	2000
`4_1600`	4	1600

Speicherbeschreibung	Speichergröße	Speichergeschwindigkeit
`4_2000`	4	2000

Wenn in diesem Fall eine Speichergröße von 2 angegeben wird, schränkt das System die möglichen Einträge für die Speichergeschwindigkeit auf 1333, 1600 und 2000 ein.

Weitere Informationen

Weitere Informationen zum Definieren von Variantentabellen finden Sie unter Definieren von Variantentabellen [Seite 44].

2.1.5.1 Syntax zur Definition einer Variantentabelle

> i Note
>
> Diese Beschreibung verwendet die folgenden Konventionen zum Darstellen der Syntaxanforderungen:
>
> Erforderliches Schlüsselwort
>
> **Optionales Schlüsselwort**
>
> **`Benutzerdefinierter Wert`**
>
> `Einander ausschließende Schlüsselwörter`

Verwenden Sie zum Definieren einer Variantentabelle die folgende Syntax:

*variantTable***`identifier`***{*

name **`„beschreibender Text 30 Byte“`** || **names** DE **`„beschreibender deutscher Text“`**

, FR **`„beschreibender französischer Text“`**

characteristics **`characteristic-id`**

--Die eingerückten Schlüsselwörter werden für jedes Merkmal in dieser Klassendefinition angegeben--

Primary

*externalTable***`'db_tablename'`**

rows 'textvalue', numericvalue;

'textvalue', numericvalue

||

file "filename.csv"

type 'CSV'

options

```
}
```

Schlüsselwort/Benutzerwert	Bedeutung	Beziehung zu anderen Schlüsselwörtern	Erforderlich/Optional
`variantTable`	Kennzeichnet den Anfang einer Variantentabellendefinition	–	Erforderlich
`identifier`	Variantentabellen-ID, die im Modell zur Referenzierung dieser Variantentabelle verwendet wird.	–	Erforderlich
`{`	Steht am Anfang der Reihe von Schlüsselwörtern, die dieses Objekt definieren	–	Erforderlich
`}`	Beendet die Objektdefinition	–	Erforderlich
`name` oder `names`	Beschreibender, sprachabhängiger Text, der als Bezeichnung für dieses Objekt verwendet wird. Wird dieser nicht verwendet, wird die ID als Name verwendet. Die vollständige Liste der verfügbaren zweistelligen Sprachen-IDs finden Sie in der kontextsensitiven Hilfe.	–	Optional
`characteristics`	Schlüsselwort, das den Anfang der Merkmalsliste für diese Variantentabelle angibt	–	Optional
Folgende Schlüsselwörter werden für jedes Merkmal angegeben			
`primary`	Kennzeichnet ein Merkmal, das zum Nachschlagen von Werten anderer Merkmale verwendet wird. Dies wird als Schlüsselfeld zum Nachschlagen von anderen Werte aus der Variantentabelle verwendet.	–	Erforderlich für mindestens ein Merkmal
Ende merkmalsspezifische Schlüsselwörter			

Schlüsselwort/Benutzer-wert	Bedeutung	Beziehung zu anderen Schlüsselwörtern	Erforderlich/Optional
`externalTable`	Wird verwendet, um dem Wissensbasis-Build-Prozess anzuzeigen, dass `variantTable` Content nicht in die Wissensbasis-Laufzeitversion eingebettet werden soll und stattdessen zur Laufzeit von der Engine aus der benannten SAP-Datenbanktabelle abgerufen werden soll.	Kann mit oder ohne die Schlüsselwörter `rows` oder `file` verwendet werden. Jedoch werden Daten, die in `rows` oder `file` definiert sind, verwendet, wenn die Variantentabellendaten ebenfalls mit dem Modell exportiert werden und keine Daten in der externen Tabelle (definiert mit dem Schlüsselwort `externalTable`) verfügbar sind.	Optional
`'db_tablename'`	Name einer Tabelle in SAP-Datenbank	Wird nur mit `externalTable` verwendet	Erforderlich
`rows(x-file)`	Wird zum Bereitstellen von Tabelleninhalt in der Tabellendefinition verwendet. Die Werte sind durch ein Komma getrennt. Zeilen werden durch ein Semikolon getrennt. Textwerte sind in Anführungszeichen eingeschlossen.	Eines der Schlüsselwörter `rows`, `file` oder `externalTable` ist erforderlich. `externalTable` kann auch zusammen mit `rows` oder `file` verwendet werden.	Optional
`file(x-rows)`	Wird zum Bereitstellen von Tabelleninhalt aus einer Datei verwendet. Die Datei muss sich in demselben Projektordner wie die Datei befinden, die diese Variantentabellendefinition enthält. Die Verwendung des .vtable-Suffix wird empfohlen.	Eines der Schlüsselwörter `rows`, `file` oder `externalTable` ist erforderlich. `externalTable` kann auch zusammen mit `rows` oder `file` verwendet werden.	Optional

Schlüsselwort/Benutzerwert	Bedeutung	Beziehung zu anderen Schlüsselwörtern	Erforderlich/Optional
`type`	Gibt den Typ der referenzierten Datei an. Mögliche Werte sind: • `vtable` (Standard): Unterstützt .vtable-Dateien mit tabstoppgetrennten Feldern ohne Anführungszeichen und ohne Escape-Zeichen • `cvs`: kommagetrennte Werte	Wird nur mit `file` verwendet	Optional

Schlüsselwort/Benutzerwert	Bedeutung	Beziehung zu anderen Schlüsselwörtern	Erforderlich/Optional
`options`	Wird zur Bereitstellung von Optionen in Bezug auf die in der Datei enthaltenen Daten verwendet. Verfügbare Optionen können im folgenden Format angegeben werden: Option1=Wert1 Option2=Wert2. CSV unterstützt beispielsweise folgende Optionen: • `charset=ISO-8859-1` Den Zeichensatz der Datei, wie z.B. „ISO-8859-1“ oder „UTF-8“. Standardmäßig Eclipse/BS-abhängig.. • `delimiter=,` Das Wertetrennzeichen. Standard ist ','. Für Tabulator- oder Leerzeichen verwenden Sie TAB oder LEER. • `quoteChar="` Das Zeichen zum Angeben eines Werts. Standard ist '"'. • `commentChars=#` Die Zeichen zum Kommentieren einer Kommentarzeile. Standardmäßig deaktiviert. • `surroundingSpacesNeedQuotes=false` Kennzeichen, das angibt, ob Leerzeichen am Anfang oder Ende einer Zelle ignoriert werden sollen, wenn sie nicht in Anführungszeichen eingeschlossen sind.	Wird nur mit `file` verwendet	Optional

Schlüsselwort/Benutzer-wert	Bedeutung	Beziehung zu anderen Schlüsselwörtern	Erforderlich/Optional
	Standardmäßig „false“, da Leerzeichen als Bestandteil eines Felds erachtet werden und gemäß RFC 4180 nicht ignoriert werden sollten. Verwenden Sie `true` zur Aktivierung. Entsprechend für `Vtable`		

Beispiel

```
variantTable VT_ERLANG {
       name "VT_ERLANG"
       characteristics
               ERLANG_LO primary,
               ERLANG_HI primary,
               BLOCKING  primary,
               NUM_LINES
       externalTable 'MY_SAP_ERLANG'
    file "VT_ERLANG.vtable"
}
```

2.1.5.2 Beispiel für eine Definition einer Variantentabelle

Kontext

Mit diesem Verfahren legen Sie Variantentabellen in der Eclipse-basierten Lösungsmodellierungsumgebung an.

Vorgehensweise

1. Wählen Sie *Datei* > *Neu* > *Variantentabelle*.
2. Geben Sie die erforderlichen Daten ein.
3. Wählen Sie *Fertigstellen*.

   Die neue Variantentabelle wird im Texteditor geöffnet.

i Note

Die Zeilen für die neue Variantentabelle müssen über den Texteditor eingegeben werden. Die Daten können innerhalb der Zeile, wie im folgenden Beispiel dargestellt, oder in eine separate Tabellendatei eingegeben werden.

Beispiel

variantTable SME_VIDEO_CARD_EXAMPLE {

name 'SME_VIDEO_CARD_EXAMPLE'

characteristics

SME_VIDEO_CARD_S **primary**,

SME_PART_NUMBER_S

rows

"LOW END", "VID1";

"HIGH END", "VID2"

}

2.1.5.3 Variantentabellensichten

Große Variantentabellen, insbesondere, wenn sie als externe Variantentabellen entworfen wurden, können aufgrund langsamerer Abfragen die Performance der Konfiguration verschlechtern. Um die Performance dieser großen Variantentabellen zu verbessern, kann der Umfang der Tabellendaten durch die Definition von Sichten für die Tabelle reduziert werden. Die für die Variantentabelle definierte Sicht gilt innerhalb einer Konfigurationssitzung.

Sie können unter Verwendung verfügbarer APIs in der Configuration Engine eine Sicht für eine Variantentabelle definieren. Sie können diese APIs mithilfe von pFunctions aufrufen. Weitere Informationen erhalten Sie in SAP-Hinweis 2509009 (Variantentabellensichten implementieren).

i Note

Durch die Definition von Variantentabellensichten wird die Performance in externen und internen Variantentabellen verbessert.

2.1.6 Dependency

Definition

Dependency is the generic term used for constraints and rules. Dependencies are defined within a solution model and specify the relationships between the characteristics and characteristic values of several classes.

Constraints

Constraints are interdependencies between objects and their characteristics. They are a primary form of declarative dependency. All declarative dependencies can be modeled using constraints. You can use them to set characteristic values or to check the consistency of assigned values. Constraints are grouped into constraint nets, which are then assigned to tasks. Constraints are used by the configurator engine to derive or infer any of the following:

- Contradiction
- Value assignment (for characteristics with an ADT, this means linking instances)
- Domain restriction
- Creation and placement of an instance in the PART_OF hierarchy (declarative selection)
- Specialization of an instance
- Relation link between instances for declared relations, when and if this feature becomes available

The configurator engine performs these derivations by applying constraint rules derived from the constraint itself.

Rules

Rules are the primary procedural form of dependency. They act as a point of reference for the classes, characteristics, materials, and variant tables created by the modeler. Rules are the procedural counterpart of constraints used when procedural logic is required, for example, counting or invoking procedural functions (pfunctions).A rule states that some action should be performed when the pattern of the rule is matched in the DDB. It can either be one of the specially provided built-in ones or an existing function (such as built-in APIs or user provided functions).Rules are grouped into rule nets, which are assigned to tasks. The tasks are then assigned to the knowledge bases.In particular, a rule can:

- Perform aggregations
- Access and modify the DDB
- Access the KB
- Trigger front-end functions
- Access and modify external data
- Make logical inferences.

More Information

For information about defining dependencies, see Defining Solution Dependencies [page 47].

2.1.6.1 Definieren von Lösungsbeziehungen

Verwendung

Mit diesem Verfahren legen Sie Lösungsbeziehungen an, indem Sie Constraints und Constraintnetze sowie Regeln und Regelnetze definieren.

Vorgehensweise

Definieren von Constraints

1. Wählen Sie *Datei* > *Neu* > *Leere Modelldatei*.
   Es wird ein Dialogfenster angezeigt, in dem Sie dazu aufgefordert werden, einen Ordner auszuwählen und einen Dateinamen für den neuen Constraint einzugeben.
2. Geben Sie die erforderlichen Daten ein.
3. Wählen Sie *Fertigstellen*.
   Es wird eine leere Datei angezeigt.
4. Drücken Sie STRG + LEERTASTE.
5. Doppelklicken Sie auf *constraint.*

   i Note

   Sie können eine Constraint-Vorlage auswählen, wenn eine geeignete verfügbar ist. Vorlagen sind durch einen grünen Punkt im Dialogfenster für die automatische Vervollständigung gekennzeichnet.
6. Geben Sie die erforderlichen Daten ein, und sichern Sie den Constraint.

Definieren von Constraintnetzen

1. Wählen Sie *Datei* > *Neu* > *Leere Modelldatei*.
   Es wird ein Dialogfenster angezeigt, in dem Sie dazu aufgefordert werden, einen Ordner auszuwählen und einen Dateinamen für das neue Constraintnetz einzugeben.
2. Geben Sie die erforderlichen Daten ein.
3. Wählen Sie *Fertigstellen*.
4. Drücken Sie STRG + LEERTASTE.
5. Doppelklicken Sie auf *constraintNet*, oder wählen Sie eine Vorlage aus.
6. Geben Sie die erforderlichen Daten ein, und sichern Sie das Constraintnetz.

Definieren von Regeln

1. Wählen Sie *Datei* > *Neu* > *Leere Modelldatei*.
   Es wird ein Dialogfenster angezeigt, in dem Sie dazu aufgefordert werden, einen Ordner auszuwählen und einen Dateinamen für die neue Regel einzugeben.
2. Geben Sie die erforderlichen Daten ein.
3. Wählen Sie *Fertigstellen*.
4. Drücken Sie STRG + LEERTASTE.
5. Doppelklicken Sie auf *rule*, oder wählen Sie eine Vorlage aus.

6. Geben Sie die erforderlichen Daten ein, und sichern Sie die Regel.

Definieren von Regelnetzen

1. Wählen Sie *Datei* > *Neu* > *Leere Modelldatei* .
   Es wird ein Dialogfenster angezeigt, in dem Sie dazu aufgefordert werden, einen Ordner auszuwählen und einen Dateinamen für das neue Regelnetz einzugeben.
2. Geben Sie die erforderlichen Daten ein.
3. Wählen Sie *Fertigstellen*.
4. Drücken Sie STRG + LEERTASTE.
5. Doppelklicken Sie auf *ruleNet*, oder wählen Sie eine Vorlage aus.
6. Geben Sie die erforderlichen Daten ein, und sichern Sie das Regelnetz.

2.1.6.2 Defining a Rule

i Note

This description uses the following conventions to illustrate the syntax requirements:

Required keyword

Optional keyword

`User-specified value`

`Mutually exclusive keyword`

Rules are used when procedural logic is required, for example, counting or invoking procedural functions (pfunctions). To define a rule, use the following syntax:

rule **`identifier`** *{*

name **`"30 bytes descriptive text"`** || **names** EN **`"descriptive English text"`**

, FR **`"descriptive French text"`**

objects: ? **`a`** **is_a (300)** **`class-id`**

where **`?b`** **=** **`adt-cstic-on-a`**

,? m **is_object (material) (300) (nr=** **`material-id`** **)**

condition:

`?a.cstic-on-a` **<|| <=|| =|| >=|| =** **`value`** **|| specified || not specified**

or || and

part_of **`(?m,?a)`**

or || and

table **`table-name (cstic-id = ?a.domain cstic-on-a)`**

> **i Note**
>
> A table statement in the “condition” section evaluates “True” if at least one row is found and “False” if no row is found.

or || **and**

...see the inline help for a full list of potential functions/content

body: then do: pfunction `pfunction-id` **(** *parameters list* **)**

||

then do: `?a.cstic-on-a ?= value || cstic`

||

then increment: `?a.cstic-on-a` **by** *i*

explanations: '`any descriptive text`'

}

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`rule`	Denotes the start of a rule definition encompassed within {...}	-	Required
`identifier`	Rule ID used in the model to reference this rule. Max. length 30 bytes	-	Required
`name` or `names`	The descriptive, language-dependent text used as a label for this object. If not used, the identifier is used as the name. See the context-sensitive help for the full list of available two-character language identifiers.	-	Optional
`objects`	Keyword indicating the start of the objects section	-	Required
`?a/?m`	A user specified symbol to represent the object. Must start with a question mark (?) but can have any alphanumeric characters after the question mark.	At least one object is required	Required

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`is_a (300)`	Keyword indicating the object is a class	-	Required
`class-id`	Class identifier of object	-	Required
`is_object (material) (300) (nr= )`	Keyword indicating the object is a material	-	Required
`material-id`	Material identifier of object	-	Required
`condition`	Keyword indicating the start of the condition. Must be a single condition statement, but can have multiple clauses joined with Boolean operators “and”/ “or”	-	Optional
`condition content`	Expressions in the condition content can include a large range of functions. Use context help (CTRL + SPACE) to view the entire list	-	Optional
`body:`	Keyword indicating the start of the body of the rule	-	Required
`then do:`	Keyword used if the rule invokes a pfunction or sets a “sticky” default value	Either `then do:` or `then increment:` is required.	Optional
`then increment:`	Keyword used if the rule increments an aggregating characteristic	Either `then do:` or `then increment:` is required.	Optional
`?a.cstic-on-a`	An aggregating characteristic for instance `?a`	Required with `then increment:`	Required with `then increment:`
`by`	Keyword denoting the range by which the increment is made	Required with `then increment:`	Required with `then increment:`
`i`	The number to be added to cstic-on-a Can be a number or the numeric result of a function.	Required with `by`	Required with `by`

Keyword/User Value	Meaning	Relation to Other Keywords	Required/Optional
`pfunction`	Keyword used to indicate that a pfunction is to be invoked	Allowed only with `then do:`	Optional
`pfunction-name`	Name of pfunction to be invoked	Required with `pfunction`	Required with `pfunction`
`parameters`	List of parameters; any specified parameter must be defined in the pfunction definition. It is not necessary to provide all parameters.	Optional	Optional
`?a.cstic-on-a`	Any characteristic for class a	-	Optional
`?=`	Indicates setting a "sticky default". Whenever the characteristic has no value assigned, this value will be assigned as its default.	Required with `pfunction`	Required with `pfunction`
`parameters`	List of parameters; must match the parameters in the pfunction definition	Required with `pfunction`	Required with `pfunction`
`explanation`	Keyword indicating the start of the explanation section.	-	Optional
`explanation text`	Text entered here is available for debugging conflicts	-	Optional

i Note

Non-sticky defaults (assigned in the class definition using the defaultValues keyword) are retained until changed by a user input or constraint. If subsequently reset to no value ("space"), the assigned default is not recovered. The cstic has no value assigned.

Example

```
rule SET_PROD_COLOR {
 objects:
  ?p is_a (300)PRODUCT
 condition:
  ?p.USAGE_TYPE = 'STEALTH'
 body:
  then do:
   pfunction SET_PROD_COLOR_PF (
    PRODUCT = ?p
    )
}
rule COUNT_COMPONENTS {
 objects:
  ?bundle is_a (300)HARDWARE_BUNDLE
      where ?comp = COMPONENTS
 body:
  then increment:
   ?bundle.COMP_COUNT by 1
    )
}
```

2.1.6.3 Defining a Constraint

> i Note
>
> This description uses the following conventions to illustrate the syntax requirements:
>
> *Required keyword*
>
> **Optional keyword**
>
> **`User-specified value`**
>
> `Mutually exclusive keyword`

Constraints assert facts and rely on deductive reasoning. To define a constraint, use the following syntax:

constraint **`identifier`** *{*

name **`"30 bytes descriptive text"`** || **names** EN **`"descriptive English text"`**

, FR **`"descriptive French text"`**

objects: ? **`a`** **is_a** (300) **`class-id`**

where **`?b`** = **`adt-cstic-on-a`**

,? m **is_object** (material) (300) (nr= **`material-id`**)

condition:

`?a.cstic-on-a` <|| <=|| =|| >=|| = *value* || **`specified`** || **`not specified`**

or || and

part_of **`(?m,?a)`**

or || and

table **`table-name (cstic-id = ?a.domain cstic-on-a)`**

or || and

...see inline help for full list of potential functions/content

restrictions:

asserted facts

i Note

For an explanation of the statements that can be used to assert facts, see the “Commands Used in the Restrictions Section of a Constraint” table below. The list, along with detailed syntax, is also available via the inline help for editing a constraint within the solution modeling environment.

inferences:

```
?a.cstic-on-a

,?m.cstic-on-a

,?a.facet cstic-on-a
```

explanations: `'any descriptive text'`

```
}
```

Keyword/User Value	Meaning	Required/Optional
`constraint`	Denotes the start of a constraint definition encompasse within {...}	Required
`identifier`	This is the constraint ID used in the model to reference this constraint. Max. length 30 bytes	Required
`name` or `names`	The descriptive, language-dependent text used as a label for this object. If not used, the identifier is used as the name. See the context-sensitive help for the full list of available two-character language identifiers.	Optional
`objects`	Keyword indicating the start of the objects section	Required
`?a/?m`	A user specified symbol to represent the object. Must start with a question mark (?) but can have any alphanumeric characters after the question mark.	Required
`is_a (300)`	Keyword indicating the object is a class	Required
`class-id`	Class identifier of object	Required
`is_object (material) (300) (nr= )`	Keyword indicating the object is a material	
`material-id`	Material identifier of object	

Keyword/User Value	Meaning	Required/Optional
`condition`	Keyword indicating the start of the condition. Must be a single condition statement, but can have multiple clauses joined with Boolean operators “and”/ “or”	Optional
`condition content`	Expressions in the condition content can include a large range of functions. Use context help (CTRL + SPACE) to view the entire list	Optional
`restrictions`	Keyword indicating the start of the restrictions section	Required
`restrictions content`	Restrictions are stated as assertions and, like condition content, can use a variety of functions. Restrictions can assert either the existence of an instance or the value of a characteristic facet	Required
`inferences`	Keyword indicating the start of the objects section	Optional
`inferences content`	Lists the asserted facts to be inferred	Optional
`explanation`	Keyword indicating the start of the explanation section	Optional
`explanation text`	Text entered here is available to help with debugging conflicts	Optional

Commands Used in the Restrictions Section of a Constraint

Command	Use
`false`	To raise a conflict, thereby making the configuration inconsistent
`find_or_create()`	To create a new instance in the configuration
`has_part()`	To add a part to a bill of material
`table()`	To look up values in a variant table
`type_of()`	To specialize a class object

Command	Use
`?class.cstic`	To assert facts about the value of a charactristic “cstic” on an instance of class “class”. `?class` is a reference to an object list in the objects section and `cstic` is a characteristic for that object.
`?class.facet cstic`	To assert facts about a facet of a characteristic “cstic” on an instance of class “class”. Examples of facets are domain, invisible, required, and noinput.
Mathematical functions	A large set of mathematical functions is available. Examples include the following functions: • rounded • sin • cos • tan • floor • log10 For a full list of the available functions, see the inline help.

Example

```
constraint PLACE_FUNIT_A_1 {
 objects:
  ?CA is_a (300)CABINET_A
 ,?FA is_a (300)FUNIT_A
 condition:
  ?FA.INST_NUM = 1
 restrictions:
  ?CA.FU_A = ?FA
 inferences:
  ?CA.FU_A
 explanations:
  "first FUNIT_A goes in CAB_A"
}
```

2.1.6.4 Beziehungsnetze

Ein Beziehungsnetz ist eine Gruppe von Constraints oder Regeln (auch als Beziehungen bezeichnet), die in einem Lösungsmodell definiert wird und die Beziehungen zwischen den Merkmalen und Merkmalswerten mehrerer Klassen angibt.

Verwendung

Constraints

Constraints sind Abhängigkeiten zwischen Objekten und deren Merkmalen. Sie werden als eine Beziehung in der Variantenkonfiguration verwendet. Sie können sie zum Festlegen von Merkmalswerten oder zum Prüfen der Konsistenz von zugeordneten Werten verwenden. In der Variantenkonfiguration können Sie Constraints für konfigurierbare Baugruppen in einer Stückliste verwenden, zwischen denen Abhängigkeiten bestehen.

Regeln

Regeln fungieren als Bezugspunkt für die vom Modeler angelegten Klassen, Merkmale, Materialien und Variantentabellen. Sie werden verwendet, wenn Verfahrenslogik erforderlich ist, beispielsweise beim Zählen oder Aufrufen von prozeduralen Funktionen (PFunctions). Regeln sind in Regelnetzen gruppiert, die Aufgaben zugeordnet werden. Die Aufgaben werden dann den Wissensbasen zugeordnet.

Struktur

Das Beziehungsnetz wird in Form einer Hierarchie im Modelldiagramm angezeigt.

Weitere Informationen

Weitere Informationen zum Definieren von Beziehungsnetzen finden Sie unter Definieren von Lösungsbeziehungen [Seite 47].

2.1.7 Benutzerdefinierte Funktionen

Benutzerdefinierte Funktionen bieten Ihnen die Möglichkeit, (komplexe) Aufgaben mithilfe von externem Quelltext auszuführen. Sie können zur Prüfung von Werten und Ableitung von Merkmalswerten verwendet werden. Sie können beispielsweise für folgende Zwecke Funktionen anlegen:

- Komplexe Berechnungen auf Basis von Merkmalswerten in der Konfiguration
- Komplexe Gültigkeitsprüfungen für zulässige Werte
- Effiziente Vorgänge unter Verwendung von Nebeneffekten, die Beschränkungen der Beziehungssyntax überwinden
- Beliebige Aufrufe an externe Programme, mit oder ohne Nebeneffekte

Benutzerdefinierte Funktionen können in Constraints und Regeln verwendet werden. In Constraints können Sie Funktionen verwenden, um die Konsistenz der eingegebenen Werte zu prüfen, beliebige komplexe Berechnungen von Eingaben zur Bestimmung von Ausgabewerten durchzuführen und Werte in externen Systemen nachzuschlagen. In Regeln können Sie Funktionen zum Auslösen der zufälligen Verarbeitung verwenden. Wenn Sie eine benutzerdefinierte Funktion in einem Constraint oder einer Regel aufrufen, verweisen Sie auf eine Java-Methode.

Verwendung

Es gibt zwei Formen von benutzerdefinierten Funktionen: deklarative Funktionen und prozedurale Funktionen („PFunctions“). Sowohl deklarative Funktionen als auch PFunctions haben dieselbe Oberfläche.

Struktur

Deklarative Funktionen und PFunctions werden unter Verwendung des folgenden Frameworks definiert:

```
function <identifier> {
  characteristics
    <parameter list>
}
```

Folgende Schlüsselwörter werden verwendet:

Schlüsselwort/Benutzerwert	Bedeutung	Erforderlich/Optional
`function`	Kennzeichnet den Anfang einer Funktionsdefinition	Erforderlich
`identifier`	Im Modell verwendete Funktions-ID zur Referenzierung dieser Funktion. Max. Länge 30 Byte. Muss dem Namen der von dieser Funktion aufgerufenen Klasse entsprechen.	Erforderlich
`{`	Steht am Anfang der Reihe von Schlüsselwörtern, die dieses Objekt definieren	Erforderlich
`}`	Beendet die Objektdefinition	Erforderlich
`characteristics`	Schlüsselwort, das den Anfang der Merkmalsliste für diese Klasse angibt	Erforderlich

Schlüsselwort/Benutzerwert	Bedeutung	Erforderlich/Optional
`parameters`	Merkmalsliste, die der Java-Klasse als Eingabe bereitgestellt wird. Alle aufgeführten Merkmale sind erforderlich, wenn diese Funktion aufgerufen wird. i Note Von einer PFunction werden keine Werte zurückgegeben.	Erforderlich
`primary`	Erforderlich für mindestens einen Parameter. Gibt für die aufgerufene Funktion an, dass der Parameter „nur Eingabe" entspricht. Es wird **nicht** angegeben, dass der Parameter erforderlich ist.	Erforderlich für mindestens einen Parameter

Weitere Informationen

2.1.7.1 Deklarative Funktionen

Deklarative Funktionen werden aus Constraints aufgerufen. Sie erhalten in ihren Argumenten Datenstrukturen, die die Ein- und Ausgabemerkmale darstellen. Unter Verwendung dieser Merkmale können sie die Werte der Eingabemerkmale in der Konfiguration lesen und Werte für die Ausgabemerkmale zurückgeben.

Eine deklarative Funktion ist von der tatsächlichen Konfiguration isoliert und kann weder den Wert eines anderen Merkmals als das beeinflussen, das als Ausgabewert an sie übergeben wird, noch andere Nebeneffekte erzeugen.

Weitere Informationen

Definieren benutzerdefinierter Funktionen [Seite 81]

2.1.7.1.1 Beispiel: Definieren von deklarativen Funktionen

Im folgenden Beispiel wird dargestellt, wie deklarative Funktionen definiert, implementiert und aufgerufen werden.

Beispiel-Framework einer deklarativen Funktion

Verwenden Sie zum Definieren einer deklarativen Funktion das folgende Modell:

```
function DETERMINE_LABEL_ID {
    characteristics
        CPU primary,
        HD primary,
        LABEL_ID
}
```

Die Funktion wird in einer Java-Klasse implementiert. Die Java-Klasse muss folgendermaßen definiert werden:

- Das Paket der Klasse muss `com.sap.sce.user` entsprechen.
- Der Klassenname muss mit dem Definitionsnamen übereinstimmen (hier: `DETERMINE_LABEL_ID`).
- Die Klasse muss die Schnittstelle `com.sap.sce.user.sce_user_fn` implementieren.

Die Schnittstelle definiert die folgende Methode, die implementiert werden muss: `boolean execute(fn_args args, Object obj);`

Die Methode hat zwei Parameter:

- `args` enthält die Abfolge von Argumenten Importargumente mit einer bestehenden Bindung und Exportargumente, für die die Bindung in der Methodenimplementierung festgelegt werden muss. Alle Merkmale, die in der Funktionsdefinition als primär angegeben wurden, werden zu Importparametern. Andere Merkmale werden zu Ausgabeparametern.
- Der Typ des zweiten Parameters `obj` ist von dem Kontext abhängig, in dem die Funktion ausgeführt wird. Wenn dieser Parameter als eine deklarative Funktion ausgeführt wird, bezieht sich der Parameter auf das Wissensbasisobjekt, das den Zugriff auf statische Informationen wie Name, Version und Profil der Wissensbasis ermöglicht.

Eine deklarative Funktion gibt einen Booleschen Wert zurück, der die erfolgreiche oder fehlgeschlagene Ausführung angibt. Eine deklarative Funktion, die im **condition**-Teil eines Constraints oder im **if**-Teil einer Einschränkung aufgerufen wird, muss einen aussagekräftigen Booleschen Wert zurückgeben, weil der Rückgabewert vom Constraint getestet wird.

Beispielimplementierung einer deklarativen Funktion

Anhand des folgenden Quelltexts wird die Implementierung einer deklarativen Funktion dargestellt.

```
public class DETERMINE_LABEL_ID implements sce_user_fn {
    public boolean execute(fn_args args, Object kbObj) {
        final sce_user_fn_logging log = new sce_user_fn_logging();

        // retrieve input characteristics
        String inputCPU = args.get_value("CPU");
        String inputHD = args.get_value("HD");

        // determine and set output characteristic
        String outputLabel = String
                .format("CPU: %s, HD: %s", inputCPU, inputHD);
        args.set_value("LABEL_ID", outputLabel);

        log.writeLogDebug(this, "Calculated label is " + outputLabel);

        return true;
    }
}
```

Bei dieser Implementierung werden die Merkmale `CPU` und `HD` aus der Konfiguration als Eingabeparameter abgerufen. Diese beiden Zeichenfolgen werden verkettet und im Ausgabeparameter `LABEL_ID` abgelegt.

Note

Die Implementierung kann nur begrenzt auf den aktuellen Status der Konfiguration zugreifen. Nur auf die explizit definierten Ein- und Ausgabeparameter kann zugegriffen werden.

Beispielaufruf einer deklarativen Funktion

Die definierte deklarative Funktion kann wie folgt in einem Constraint verwendet werden:

```
function DETERMINE_LABEL_ID {
        CPU      = ?PC.CPU,
        HD       = ?PC.HD,
        LABEL_ID = ?PC.LABEL_ID
}
```

Die Merkmale auf der linken Seite sind Merkmale der Funktion. Die Merkmale auf der rechten Seite sind Merkmale der Instanz (referenziert mit `?PC`) der Konfiguration.

2.1.7.2 PFunction

Eine „PFunction“ (prozedurale Funktion) ermöglicht den Lese- und Schreibzugriff auf die Konfiguration und dynamische Datenbank (im Gegensatz zu einer deklarativen Funktion, die einfach die Wissensbasis liest, um Exportparameter abzuleiten). PFunctions können nur in Regeln verwendet werden, richten sich aber an alle Konfigurationsobjekte.

PFunctions nehmen alle Änderungen als Nebeneffekte vor und sind nicht deklarativ. Vom TMS werden keine Aktionen verfolgt (es sei denn, die Nachverfolgung wird manuell implementiert). Im Gegensatz zu deklarativen Funktionen können PFunctions die Konfiguration direkt ändern. Eine PFunction kann die Ausführung von Beziehungen über das API „Prüfen“ explizit aufrufen.

Weitere Informationen

Definieren benutzerdefinierter Funktionen [Seite 81]

Beispiel: Definieren von PFunctions [Seite 61]

2.1.7.2.1 Testen von PFunctions

Voraussetzungen

- Sie haben eine Wissensbasis angelegt.
- Sie haben eine Startkonfiguration angelegt (siehe Anlegen einer Startkonfiguration [Seite 81]).

Kontext

Wenn Sie eine Wissensbasis zum Testen öffnen, werden PFunctions möglicherweise nicht erwartungsgemäß ausgeführt. In diesem Fall können Sie dieses Verfahren zum Testen und Debuggen der PFunctions verwenden.

Vorgehensweise

1. Öffnen Sie die Perspektive *Debug* und wählen Sie ▶ *Ausführen* ▶ *Debug-Konfigurationen...* ◀.
2. Klappen Sie den Knoten *Eclipse-Anwendungen* auf.
3. Wählen Sie eine Startkonfiguration.
4. Wählen Sie die Drucktaste *Debug*.

   Die Test-Benutzungsoberfläche wird in einem neuen Fenster angezeigt. Anschließend können Sie aus der *Debug*-Perspektive in der Sitzung der Lösungsmodellierungsumgebung Breakpoints im Java-Programm festlegen.

2.1.7.2.2 Beispiel: Definieren von PFunctions

Im folgenden Beispiel wird dargestellt, wie PFunctions definiert, implementiert und aufgerufen werden.

Beispiel-Framework einer PFunction

Verwenden Sie zum Definieren einer PFunction das folgende Modell:

```
function NUMBER_OF_ITEMS {
    characteristics
        PC_REF primary,
        HOLDS_HDS primary,
}
```

Die Funktion wird in einer Java-Klasse implementiert. Die Java-Klasse muss folgendermaßen definiert sein:

- Das Paket der Klasse muss `com.sap.sce.user` entsprechen.
- Der Klassenname muss mit dem Definitionsnamen übereinstimmen (hier: `NUMBER_OF_ITEMS`).
- Die Klasse muss die Schnittstelle `com.sap.sce.user.sce_user_fn` implementieren.

Die Schnittstelle definiert die folgende Methode, die implementiert werden muss: `boolean execute(fn_args args, Object obj);`

Die Methode hat zwei Parameter:

- `args` enthält die Abfolge von Argumenten Importargumente mit einer bestehenden Bindung und Exportargumente, für die die Bindung in der Methodenimplementierung festgelegt werden muss. Alle Merkmale, die in der Funktionsdefinition als primär angegeben wurden, werden zu Importparametern. Andere Merkmale werden zu Ausgabeparametern.
- Der Typ des zweiten Parameters `obj` ist von dem Kontext abhängig, in dem die Funktion ausgeführt wird. Wenn dieser Parameter als eine PFunction ausgeführt wird, bezieht sich der Parameter auf die Konfiguration selbst, wodurch Änderungen direkt in der Konfiguration vorgenommen werden können.

Der Boolesche Rückgabeparameter kann im Kontext von PFunctions nicht verwendet werden. Während der Rückgabewert einer deklarativen Funktion eine Bedeutung übermittelt, sollte der Rückgabewert einer PFunction immer True entsprechen. Wenn der Rückgabewert False entspricht, kann dies zu Fehlern in der Engine und einer inkonsistenten Konfiguration führen.

i Note

Eine Funktionsimplementierung (d.h. die Java-Klasse) kann als Test oder zum Festlegen von cstic-Werten aufgerufen werden und kann ggf. sogar als Implementierung einer PFunction verwendet werden. Diese Mehrfachverwendung macht eine sehr sorgfältige Implementierung erforderlich. Sie ist zulässig, wird jedoch nicht unbedingt empfohlen.

Beispielimplementierung einer PFunction

Anhand des folgenden Quelltexts wird die Implementierung einer PFunction dargestellt.

```
public class NUMBER_OF_ITEMS implements sce_user_fn {

  public boolean execute(fn_args args, Object configObj) {
    // retrieve input characteristic PC_REF. PC_REF is an ADT
    // characteristic and therefore can be converted to an ddb_inst
    // object
    ddb_inst instance = (ddb_inst) args.find("PC_REF").kb_get_binding();
```

```
     try {
       // retrieve the values of characteristic HOLDS_HDS
     kb_type instType = instance.ddb_get_inst_type();
     kb_cstic csticToCnt = instType.kb_has_cstic_p("HOLDS_HDS");
     read_only_sequence rs = instance.ddb_get_values(csticToCnt);

     // set the value of characteristic NUMBER_OF_HDS
     instance.ddb_set_or_replace_val(
         instType.kb_has_cstic_p("NUMBER_OF_HDS"),
         float_value_imp.get_float_value(rs.length()),
         ((cfg_imp) configObj).tms_get_generic_default_owner());

     } catch (Exception e) {
       final sce_user_fn_logging log = new sce_user_fn_logging();
       log.writeLogError(this, e.getMessage());
     }

     return true;
   }
 }
```

Diese Implementierung zählt die Anzahl der dem Merkmal `HOLDS_HDS` zugeordneten Werte. Die abgerufene Zahl wird dem Merkmal `NUMBER_OF_HDS` zugeordnet.

Die Implementierung kann auf den gesamten Status der Konfiguration zugreifen. Argument `configObj` ist eine Instanz der Klasse `cfg_imp`, die verwendet werden kann, um den Status der Konfiguration auf nicht deklarative, verfahrensorientierte Art zu ändern. In dem Beispiel wird das Merkmal `NUMBER_OF_HDS` geändert, das nicht Bestandteil der Argumentliste der PFunction ist.

Die Klasse `com.sap.sce.user.scelib` stellt Hilfsmethoden bereit, die von PFunctions zum Lesen aus und Schreiben in die aktuelle Konfiguration verwendet werden können. Diese Klasse stellt u.a. die folgenden Funktionen zur Verfügung:

Methode	Zweck
`scelib.get_value(inst, csticName)`	Liest zugeordneten Wert von `<csticName>`
`scelib.get_values(inst, csticName)`	Liest zugeordnete Werte des mehrwertigen Merkmals `<csticName>`
`scelib.set_value(inst, csticName, val)`	Legt den Wert `<val>` auf `<csticName>` fest
`scelib.vt_select_buffered(kBase, match, VTabName, condition, order)`	Wählt Zeilen der Variantentabelle `<VTabName>`, die die `<condition>` befolgen, sortiert nach `<order>`
`scelib.get_domain(inst, csticName)`	Liest aktuelle Domäne von `<csticName>`
`scelib.restrict_dom(inst, csticName, dom, owner)`	Schränkt die Domäne von `<csticName>` durch den Zeichenfolgen-Array `<dom>` ein

Beispielaufruf einer PFunction

PFunctions werden auf dieselbe Art und Weise wie deklarative Funktionen aufgerufen, z.B.:

```
function NUMBER_OF_ITEMS {
        PC_REF = ?PC.PC_REF
}
```

PFunctions können nur im Regelrumpf aufgerufen werden.

2.1.8 Oberflächendesign

Ein Oberflächendesign ist ein Objekt, das Merkmalsgruppen definiert und diesen Gruppen Merkmale zuordnet. Die Konfigurationsbenutzungsoberfläche verwendet diese Informationen zur Darstellung von Merkmalen auf dem Bildschirm. Oberflächendesignobjekte können mit Klassen oder Materialien verknüpft sein und ermöglichen so unterschiedliche Darstellungen der Merkmale auf Grundlage der Klasse oder des Materials.

Das Oberflächendesign wird von Unterklassen übernommen und kann von übergeordneten Klassen überschrieben werden. Wenn eine Unterklasse oder ein Material von mehreren übergeordneten Klassen mit einem Oberflächendesign erweitert wird (dieses übernimmt), **muss** die Unterklasse ihr eigenes Oberflächendesign definieren.

2.1.8.1 Defining an Interface Design

i Note

This description uses the following conventions to illustrate the syntax requirements:

Required keyword

Optional keyword

`User-specified value`

`Mutually exclusive keyword`

To define an interface design, use the following syntax:

interfaceDesign **`identifier-1`** *{*

group **`identifier-A`** **{**

characteristics

`characteristic-id, characteristic-id,`

urls { **`"url"`** *label* **`"30 bytes descriptive text"`** , **`"url"`** *label* **`"30 bytes descriptive text"...`** *}*

}

group **`identifier-B`** **{**

characteristics

`characteristic-id, characteristic-id,`

urls { **`"url"`** *label* **`"30 bytes descriptive text"`** , **`"url"`** *label* **`"30 bytes descriptive text"...`** *}*

```
}
}
```

Keyword/User Value	Meaning	Required/Optional
`interfaceDesign`	Denotes the start of an interface design object encompassed within {...}	Required
`identifier-1`	This is the interface design ID used in the model to reference this interface design. Max. length 18 bytes.	Required
`group`	Denotes the start of a group definition	Required
`identifier-A`	This is the group ID, the technical name of the group of characteristics. Max. length 18 bytes.	Required
`name`	This is the group name used to name/label the grouping on the UI. For example, if the UI presents groups as tabs, this is the tab label if no name is provided. Max. length 30 bytes.	Optional
`characteristics`	Indicates the start of the list of characteristics in this group	Required
`cstic-1`	A valid characteristic name	Required
`urls`	Keyword associating a URL with this group. URLs are not accessible via the engine API. We suggest that you do not use them in this release.	Optional
`label`	A label for the URL	Optional

2.1.9 Stücklisten

Eine Stückliste ist eine Liste von Materialien, Baugruppen, Komponenten und Teilen, die zur Herstellung eines Produkts benötigt werden. Sie kann zur Definition einfacher Beziehungen verwendet werden, bei denen eine Komponente mehrere Teilkomponenten umfasst. Falls erforderlich, können Sie eine maximale und eine minimale Menge für jedes Material und jede Klasse in der Stückliste definieren.

Weitere Informationen

Weitere Informationen zum Definieren von Stücklisten finden Sie unter Definieren dynamischer Stücklisten [Seite 66].

2.1.9.1 Definieren dynamischer Stücklisten

Kontext

Mit diesem Verfahren legen Sie neue Stücklisten in der Eclipse-basierten Lösungsmodellierungsumgebung an.

Vorgehensweise

1. Wählen Sie *Datei* > *Neu* > *Leere Modelldatei*

   Es wird ein Dialogfenster angezeigt, in dem Sie dazu aufgefordert werden, einen Ordner auszuwählen und einen Dateinamen für die neue Stückliste einzugeben.
2. Geben Sie die erforderlichen Daten ein.
3. Wählen Sie *Fertigstellen*.

   Im Texteditor wird eine leere Datei angezeigt.
4. Drücken Sie `STRG` + `LEERTASTE`, und doppelklicken Sie auf *bom*.

   > i Note
   >
   > Sie können auch **bom** eingeben.
5. Geben Sie die erforderlichen Daten ein, und sichern Sie die neue Stückliste.

   > i Note
   >
   > Optional können Sie einen Mindest- und einen Höchstwert für jedes Material oder jeder Klasse angeben.

Beispiel

Die nachfolgend dargestellte Stückliste umfasst eine Klasse und zwei Materialien:

```
bom SME_WORKPLACE {
10 class SME_COMPUTER min 0 max 9999,
20 material SME_DESKTOP min 0 max 9999,
30 material SME_DESKTOP min 0 max 9999
}
```

2.1.10 Model Syntax and Logical Validations

SME not only performs language syntax validation but extends it further to also do logical validations. These logical validations are performed at compile time which helps in providing feedback during the model development phase itself, thus avoiding the expensive test cycle process. These logical validations include:

- **Using default assignment operator “?=” is not allowed in constraint restriction.**
  The default assignment can be used in rules for assigning the default values, however, the same is not true for constraints. If it used in constraints, an error is thrown.
- **Warning for free variables in constraint declaration.**
  A free variable in a constraint is defined in the object section of a constraint, but is never used in the condition or restriction section. Such variables have an impact on the execution of the constraint. A warning message is shown in such scenarios. Refer to SAP Note 2877744 for more details.
- **Validation check in find_or_create**
  If find_or_create is used in the restriction section using the‘with’ argument, SME does a logical check if the characteristics used in the ‘with’ argument are defined in a class, else a validation error is shown.
- **Validation check on assigning domain in constraint restriction**
  In a constraint, if a characteristic domain is assigned in the restriction section, then that characteristic should be of restrictable type ,else an error is shown.

2.1.11 Musterlösungsmodelle

Die Lösungsmodellierungsumgebung wird mit einigen Musterlösungsmodellen ausgeliefert, die Ihnen bei der Erstellung produktiver Lösungsmodelle als Referenz dienen.

Weitere Informationen über das Anlegen produktiver Projekte auf Basis dieser Muster finden Sie über den nachfolgenden Link.

Related Information

Creating a Solution Sales Configuration Project [Seite 77]

2.1.11.1 KBO with MCI

SAP Solution Sales Configuration supports the application of predetermined rules to orchestrate between knowledge bases with multi-configuration instances (MCI). This sample model provides insight into how multi configuration instances can be created and managed across multiple configurations through the knowledge base orchestration process.

For more information about this, refer to SAP Note 2550572 (SAP SSC Knowledge Base Orchestration (KBO) and Multi Configuration Instances (MCI)).

2.1.11.2 FBS_SSC_CA

Ein Modell zur Darstellung, wie Hardware, Software und Services zusammen paketiert werden können und wie das Guided Selling aktiviert werden kann.

Das Modell FBS_SSC_CA dient der Konfiguration eines Datenzentrums und der entsprechenden Supply Server, wodurch eine Anwendung mit verschiedenen Benutzertypen unterstützt wird. Auf Grundlage der angegebenen Anzahl je Benutzertyp erzeugt das Modell die Anzahl der erforderlichen Server mit entsprechendem Speicher, CPU-Geschwindigkeit und Anzahl an CPUs.

Darüber hinaus ermöglicht das Modell dem Benutzer die Übersteuerung empfohlener Werte.

Schlüsselwörter und Schlüsselkonzepte

Schlüsselwörter	Beschreibung
Zählen	Werte zusammenzählen, zählen und kumulieren
ADT	Verwendung abstrakter Datentyp-Zeiger zur Darstellung von Beziehungen zwischen Lösungskomponenten
Nichtteil-Instanzen	Hinzufügen von Inhalten ohne Verwendung einer Stückliste
Guided Selling	Hier wird das rudimentäre Guided Selling erläutert

Übersteuern von Standardwerten im System

1. Geben Sie die Anzahl an Benutzern für einen oder mehrere Benutzertyp(en) an.
   Im Modell wird eine Liste der potenziellen Server gemäß den Anforderungen des relevanten Benutzertyps bereitgestellt.
2. Wählen Sie den relevanten Servernamen aus.
   Das Modell erzeugt die erforderliche Anzahl an Servern.
   Bis zu 14 einzelne Server sind in einem Blade-System enthalten. Jeder einzelne Server ist mit dem entsprechenden Speicher und den entsprechenden CPUs konfiguriert. Sie geben einfach nur die Anzahl an Benutzern ein und treffen Ihre Auswahl aus einer Reihe von potenziellen Servern. Das Modell führt den Prozess automatisch aus. Gegebenenfalls können Sie außerdem das Standardsystemverhalten übersteuern.
   Weitere Informationen finden Sie unter **Speicherübersteuerung**.

3. Wählen Sie zusätzliche Software und/oder Services aus.
   Diese können mit den Servern verknüpft werden, indem die im Merkmal *Wird ausgeführt auf HW* in Softwareinstanzen oder im Merkmal *Stellt bereit* in Serviceinstanzen aufgeführten Server ausgewählt werden. Die Verknüpfung kann auch von den Hardwareinstanzen aus erfolgen, indem Sie die Merkmale *Führt SW aus* und *Bereitgestellt durch* verwenden.
   Das Modell aktualisiert die Belegpositionsmenge für die Software- und Serviceinstanzen auf Basis der Anzahl der damit verknüpften Server. Wenn die Software beispielsweise auf 15 Servern ausgeführt wird, wird die Belegpositionsmenge auf 15 festgelegt. Dieser Prozess wird von dem Modell gesteuert und kann auch anders konfiguriert werden.
   Der Benutzer kann die Speicherwerte in jedem Blade-System übersteuern, indem er das Merkmal *SPEC_MEM* vom Systemstandard „System“ auf „Benutzer“ festlegt. Ist es auf „Benutzer“ festgelegt, ist das Speicherfeld für ein Update verfügbar. Sie können einen neuen Wert angeben, um diesen Speicher auf alle Server in diesem spezifischen Blade-System zu verteilen. Dies hat keine Auswirkung auf andere Blade-Systeme in der Konfiguration.

Übersteuern von anderem Standardverhalten

Das Modell wurde so definiert, dass automatisch alle Software und Services mit allen Serverinstanzen verknüpft werden. Dieses Verhalten des Modells kann übersteuert werden, indem das Merkmal *Automatische Zuordnung* in einer der folgenden Instanzen geändert wird:

- Server
- Software
- Service

Wenn die Einstellungen geändert werden, werden Zuordnungen zurückgenommen und können manuell neu zugeordnet werden.

Related Information

Beschreibung von Modellierungsverfahren [Seite 69]

2.1.11.2.1 Beschreibung von Modellierungsverfahren

Related Information

Guided-Selling-Fragebogen [Seite 70]
Auswählen von Servern [Seite 70]
Hinzufügen von Software und Services [Seite 72]

Verknüpfen von Services, Hardware und Software [Seite 72]
Zählen und Festlegen von Belegpositionsmengen [Seite 73]

2.1.11.2.1.1 Guided-Selling-Fragebogen

Die Wurzelinstanz der Konfiguration umfasst eine Reihe von Eingabeaufforderungen/Fragen, über die Sie Ihre Unternehmensanforderungen angeben können. In diesem Fall sammelt eine einzelne Frage zum Benutzervolumen (Anzahl an Benutzern je Benutzertyp) alle erforderlichen Informationen für das Modell, auf deren Grundlage das Modell die Typen der geeigneten Server ermittelt.

Die Benutzereingabe für diese Frage wird in die folgenden Felder eingetragen:

- `STD_SELFSERV_USERS`
- `SCM_SALES_USERS`

Da beide Felder auf ähnliche Art und Weise funktionieren, finden Sie weitere Informationen in den Eigenschaften von `STD_SELFSERV_USERS`.

Klicken Sie mit der rechten Maustaste auf das Feld und wählen Sie *Referenzen suchen*, um die Einschränkungen zu suchen, in denen es verwendet wird:

- `DEMAND_STD_SELF`
  Hier wird das Feld `STD_SELFSERV_USERS` zur Berechnung des *Bedarfs* verwendet, d.h. die Anzahl an CPUs, CPU-Geschwindigkeit und RAM, die erforderlich sind, um den Typ und die Anzahl der angegebenen Benutzer zu unterstützen.
- `USER_MASTER_DATA`
  Hier wird das Feld `STD_SELFSERV_USERS` verwendet, um eine Instanz für spezifische Daten zu einem Benutzertyp in der Konfiguration anzulegen. In dieser Einschränkung sehen Sie den Befehl `find_or_create` und können mit der rechten Maustaste auf den Klassennamen `USER_MASTER_DATA` klicken, um Verweise darauf zu suchen. Die Einschränkung `GET_USER_MASTER_DATA` ruft Daten aus einer Variantentabelle ab und speichert sie in der Instanz `USER_MASTER_DATA`.
  Somit arbeiten die zwei Einschränkungen zusammen, um Daten für einen bestimmten Benutzertyp abzurufen, für den Benutzer angegeben wurden. In anderen Einschränkungen wird, wenn benutzerspezifische Daten erforderlich sind, auf die Instanz `User_Master_Data` verwiesen.
- `VISIBLE_INST`
  Hier wird das Feld `STD_SELFSERV_USERS` verwendet, um die Sichtbarkeit der zwei anderen Felder zu steuern. Wird kein Wert angegeben, wird die Liste der Servernamen zusammen mit der Liste der Zeiger auf Server, die der Konfiguration bereits hinzugefügt wurden, ausgeblendet.

2.1.11.2.1.2 Auswählen von Servern

Das Feld, das die Liste der Server enthält, wird angezeigt, nachdem die Anzahl an Benutzern in eines oder beide der Felder *Anzahl von Benutzern* eingetragen wurde. Diese Wertelisten werden in Form von Variantentabellen gespeichert, und der Zugriff darauf erfolgt anhand von Regeln. Dies wird nachfolgend ausführlicher erläutert:

`STD_SELF_SERVER_NAME` ist das Auswahlfeld für den Servernamen und wird durch die folgenden Einschränkungen beschränkt.

- `VISIBLE_INST`
  Diese Einschränkung wird im Kapitel **Guided-Selling-Fragebogen** erörtert.
- `SERVER_MASTER_DATA`
  Diese Einschränkung verwendet dieselben Techniken, die für `USER_DATA` verwendet werden.
- `DMN_SERVER_AND_CPU_SPD`
  Diese Einschränkung beschränkt die Domäne der Werte von *Cpu_Speed* und *Server_Name*, indem ein Feld verwendet wird, das in der Einschränkung `DEMAND_STD_SELF` auf Basis von *User_type* berechnet wurde. Nur die CPU-Geschwindigkeiten, deren Werte größer oder gleich der für den Benutzertyp erforderlichen Geschwindigkeit sind:

  ```
  ?S.domain ACTL_STD_SELF_CPU_SPD >= ?S.REQ_STD_SELF_CPU_SPD
  ```

  Die Einschränkung verwendet weiterhin die beschränkte Domäne, um die zulässigen Servernamenwerte unter Verwendung einer Variantentabelle einzuschränken:

  ```
  table T_SERVER_DATA
                (SERVER_NAME = ?S.domain STD_SELF_SERVER_NAME
                 ,PROCESSOR_SPD = ?S.domain ACTL_STD_SELF_CPU_SPD
                        )
  ```
- `COMPUTE_REQ_SERVERS`
  Diese Einschränkung berechnet die Anzahl an Servern, die für den Speicherbedarf benötigt wird, sowie die erforderliche Anzahl an CPUs. Dann werden diese Werte verglichen und die Anzahl an Servern wird als der größere dieser beiden Werte abgerufen.
- `STD_SELF_INST_1` und `STD_SELF_INST_N`
  Diese Einschränkungen legen zusammen die erforderliche Anzahl an Servern an, wie oben dargestellt. Folgende Konditionen werden bei den Berechnungen von den Einschränkungen berücksichtigt:
  - `STD_SELF_INST_1`: Wenn es wahr ist, dass mindestens 1 (mehr als 0) Server erforderlich ist, dann ist es auch wahr, dass es einen Server mit der Instanznummer1 gibt.
  - `STD_SELF_INST_N`: Wenn es wahr ist, dass viele Server erforderlich sind und ein Server mit einer Instanznummer < die erforderliche Nummer vorhanden ist, ist es auch wahr, dass ein Server mit Instanznummer 1 größer als der bestehende ist.
    Um dies darzustellen, gehen wir davon aus, dass 3 Server erforderlich sind.
    1. `STD_SELF_INST_1` legt eine Serverinstanz mit `SSC_INSTANCE_NUM` = 1 an.
    2. `STD_SELF_INST_N` wird ausgeführt, wenn das zugehörige Muster der Objekte übereinstimmt.
    3. Die Anzahl der erforderlichen Server ist 3 und es ist eine Serverinstanz mit `instance_num` =1 (die < 3 entspricht) vorhanden.
    4. Bestätigt, dass eine weitere Serverinstanz mit `SSC_INSTANCE_NUM` um eins größer als die gefundene ist.
    5. Diese Einschränkung wird für jede neu angelegte Serverinstanz ausgeführt `instance_num` = 3 (nur für Instanzen mit `instance_num` < 3).

Related Information

Guided-Selling-Fragebogen [Seite 70]

2.1.11.2.1.3 Hinzufügen von Software und Services

Software wird hinzugefügt, indem Werte im Feld *SOFTWARE_SELECT* ausgewählt werden und Services werden hinzugefügt, indem Werte in *INST_SERVICE_SELECT* (Installationsservices) oder *MAINT_SERVICE_SELECT* (Wartungsservices) ausgewählt werden.

Bei diesen Feldern handelt es sich um Mehrwertfelder.

Die Einschränkung `CREATE_SERVICE_INST` fügt Serviceinstanzen hinzu und wird für jeden Wert im Mehrwertfeld einmal ausgeführt. Durch Verwendung der Werte im Feld zum Festlegen eines Merkmalswerts in der Serviceinstanz wird sichergestellt, dass jede Instanz eindeutig ist.

```
constraint CREATE_SERVICE_INST {
          objects:
          ?S is_a (300) FBS_SSC_CA
         condition:
          ?S.SERVICE_SELECT specified
         restrictions:
           find_or_create
            ((300) SERVICE,
             with SERVICE_PROFILE = ?S.SERVICE_SELECT;
                 IS_PART_OF_SD_SOFT = ?S;
                 SERVICE_IN_SOL_ADT = ?S)
         explanations:
           "CREATE service instance. multiValue will find_create for each value."
        }
```

Note

Wenn `SERVICE_PROFILE` für die Serviceinstanz in der Anweisung `find_or_create` nicht festgelegt ist. In diesem Fall würde nur eine Serviceinstanz angelegt und in der Rechtfertigungsverwaltung der Engine einmal für jeden Wert im Feld *SERVICE_SELECT* gerechtfertigt werden.

Eine ähnliche Technik wird zum Hinzufügen von Softwareinstanzen verwendet.

2.1.11.2.1.4 Verknüpfen von Services, Hardware und Software

Während jede zur Lösung hinzugefügte Position eine Komponente der Lösung darstellt, bestehen andere Beziehungen zwischen diesen Komponenten. Diese Beziehungen werden mithilfe von ADT ausgedrückt, die als Zeiger auf andere Instanzen fungieren und durch folgende Einschränkungen beschränkt werden:

- LINK_SW_HW, LINK_HW_SW
- LINK_HW_SV, LINK_SV_HW
- LINK_SV_SW LINK_SW_SV
  Jede der Einschränkungen „LINK_...“ stellt sicher, dass wenn Position A auf Position B zeigt, Position B auch auf Position A zeigt. Beispielsweise stellt `LINK_SW_HW` sicher, dass wenn eine Software auf einem Server ausgeführt wird, der Server auch diese Software ausführt.
- AUTO_ASSIGN_SW und AUTO_ASSIGN_SV
  Diese Einschränkungen verwenden einen Switch (`AUTO_ASSIGN_TXT`) um zu bestimmen, ob das System diese Beziehungen automatisch herstellen soll, ob es also davon ausgehen kann, dass jegliche Software auf jeglicher Hardware ausgeführt wird, jegliche Software und Hardware von allen Services in der

Konfiguration „bereitgestellt wird“ oder ob keine solchen Beziehungen vorausgesetzt werden. Werden keine Beziehungen vorausgesetzt, kann der Benutzer die Beziehungen manuell zuordnen, indem er in den ADT-Merkmalsfeldern die entsprechende Auswahl trifft.

2.1.11.2.1.5 Zählen und Festlegen von Belegpositionsmengen

Das Modell umfasst ein Referenzmerkmal, mit dem es möglich ist, im Modell auf im Verkaufsbeleg enthaltene Informationen zu verweisen, z.B. Kundennummer, Land, Verkaufsorganisation usw. Außerdem können im Modell Werte von Feldern im Verkaufsbeleg, wie etwa die Belegpositionsmenge, festgelegt werden. Das Referenzmerkmal in diesem Modell, `STOP_MENGE`, wird zum Festlegen der Belegpositionsmenge verwendet.

```
characteristic STPO_MENGE {
            names
                  EN 'Component quantity',
                  DE 'Komponentenmenge'
            numericLength 13 decimalPlaces 3
            negativeValues
            reference table 'STPO' field 'MENGE'
        }
```

Um dies zu veranschaulichen, nehmen wir an, dass die Menge einer Software von der Anzahl an Servern abhängig ist, auf der sie ausgeführt wird.

Wenn 1 Server die Software ausführt, entspricht die Positionsmenge 1; wenn sie jedoch auf 10 Servern ausgeführt wird, entspricht die Menge 10. Eine ähnliche Logik gilt für Servicepositionen.

Weitere Informationen dazu, wie die Anzahl an Softwarelizenzen gezählt wird, finden Sie in Regel `AGGR_SW_LIC` und Einschränkung `SET_SOFTWARE_STPO_MENGE`; dort wird erläutert, wie dieser Wert dem Referenzmerkmal zugeordnet wird.

2.2 Setup of Solution Modeling Environment

Use

You use this process to define a solution model in the Eclipse-based solution modeling environment, test it, and transport it to a target system for use in the sales ordering process.

You perform these steps in Eclipse, using the *SAP Modeling* perspective. To change to a different perspective, choose *Window* > *Open Perspective* > *Other...*

Note

- During SME installation, the system on which Eclipse is running must have internet connectivity. Eclipse uses the default internet connectivity configured by proxy settings on OS level (for example, in *Windows* > *Internet Options* > *Connections* > *LAN Settings* or any other OS connectivity settings). If no internet connectivity is configured, you must configure the proxy settings in Eclipse under

Windows > *Preferences* > *General* > *Network Connection*, as part of the standard Eclipse set up procedure.

- If you are working as part of a team and are using a source code management system, you must first check out a file for editing, and then check it in after completing editing.
  For more information, see chapter **Collaboration Between Modelers**.

Prerequisites

- You have configured the connection to the target system.
- You have imported the configuration master data (see Importing Configuration Master Data in the Solution Modeling Environment [page 104]).
- You have maintained your model templates (see Maintaining Modeling Templates [page 112]).

Process

1. You create or open the project for the solution model (see Creating Model Projects [page 77]).
2. You define the model master data, including characteristics and classes (see Defining Characteristics [page 25] and Defining Classes [page 15]).
3. You define materials (see Defining Materials [page 20]).
4. You define variant tables (see Defining Variant Tables [page 44]).
5. You define dynamic bills of material (BOMs) (see Defining Dynamic Bills of Material [page 66]).
6. You define dependencies and dependency nets (see Defining Solution Dependencies [page 47]).
7. You define a knowledge base (see Defining Knowledge Bases [page 10]).
8. You test the model locally (see Testing Models Locally [page 93]).
9. You export the model to the target system (See Exporting a Knowledge Base Runtime Version [page 86]).

Result

Your solution model is available for solution configuration in the target system.

Related Information

Zusammenarbeit zwischen Modelern [page 113]

2.2.1 SAP Modeling Perspective

The SAP Modeling perspective features a number of views that allow you to represent your solution model in different ways:

Component	Description
Project Explorer	The Project Explorer view allows you to explore the project as it exists in the file system, that is, as a set of folders and files. New files can be added, changed, or deleted just as they are in Windows Explorer. Updates here are also made in the local workspace. Adding a folder here, for example, adds a folder on the local disk in your workplace and deleting files removes them from the local disk. The operations available are shown in the menus or by right-clicking an object (context menu) in the Project Explorer. All operations, with the exception of those in the *SAP Modeling* menu, are standard Eclipse functions. For more information, see the Eclipse documentation.
Model Explorer	The Model Explorer view allows you to explore the project as it is seen by SAP Solution Sales Configuration, that is, as a set of objects - characteristics, classes, constraints, and so on. **Note** Objects can be opened and edited from this view, and also created in existing files. New *.ssc files can be created in the *Project Explorer* view only.

Component	Description
Model Graph	The Model Graph view shows the relationship between classes or dependencies in a graphical view. It is useful for understanding the relationships among classes and materials, or between dependencies. To view the class hierarchy, use the donut icon; to view the materials, click the hierarchy icon; to view the dependency structure, click the cube icon. When you set *Link to Editor*, the editor repositions to the definition of the object selected in the graph and the graph highlights the object based on the object definition being edited. The scope of objects shown in the graph can be controlled in two ways. First, in the configuration menu (which is accessible using the upside down triangle icon), choose *Workspace* to see objects from all open projects in your workspace, or choose *Selected Project* to see only objects in the project currently selected. You can also choose *Selected Project (Incl. References)* to see objects in the selected project and all of its open reference projects. The second way to control the scope of display in the Model Graph view is to right-click any of the displayed objects and set a filter. Setting a filter limits the view to this object and any subordinate objects. **Note** Objects can be opened and edited from this view, but not added or deleted. If you open the file in which an object is defined and set *Link to Editor*, the view will reposition to the definition of the object selected in the graph, and the graph will reposition (highlight) to the object based on the object definition being edited.
Problems	The Problems view shows messages from invoked functions, such as *Export Knowledge Base* or *Validate Model*. Double-click a message to go to the error location.

Component	Description
Search	The Search view shows the results of a search. You can also initiate a search from the Search view or the *Search* menu. The yellow up and down arrows locate the next or previous match; the plus (+) and minus (−) icons expand or collapse all entries in the results tree. The refresh icon reruns the current search, and the search history icon shows a list of previous searches, which can be selected and rerun. The downward triangle icon can be used to change the view from a tree to a list, set a filter, and update general search preferences through a link.
Outline	The Outline view works in tandem with the editor. It shows all the objects defined within the active file in the editor. You can also sort the object definitions in this view.

2.2.1.1 Creating Model Projects

Use

You use this procedure to create a new SAP Solution Sales Configuration model project in the Eclipse-based solution modeling environment.

Procedure

i Note

Creating an empty model file is a quick way to start a new object definition. You can then use the context help to complete the definition.

1. Choose *File* > *New* > *SAP Solution Sales Configuration Model Project* .
2. Enter the project name and location.
3. Choose *Finish*.
   The new project is displayed in the *Project Explorer*.

2.2.1.1.1 Creating a Solution Sales Configuration Project

You use this procedure to create a solution configuration model from an example in the Eclipse-based solution modeling environment

Creating an SSC Model Project

1. Choose *File* > *New* > *Other*
2. In the *Select a Wizard* window, expand the *SAP Solution Sales Configuration* node and select *SAP Solution Sales Configuration Model Project*
3. Choose *Next*
4. In the next window, enter the project name and a location for the project
5. Choose *Finish*

Result

You can see the new project in the *Project Explorer*.

Creating an Example SSC Model Project

1. Choose *File* > *New* > *Other*
2. In the *Select a wizard* window, select *Examples*
   From the options, select the sample project you want to include in the workspace and click on *Next*
3. Click on *Finish*

Result

You can see the sample project in the *Project Explorer*.

> **i Note**
>
> It is recommended to organize the project files in an object-oriented way. For example, you can have one file with the classes of one (or several related) objects and their cstics. Similarly, you can have another file which stores the constraints and constraint nets (of the completed model, or separated into several object-oriented files). Similarly, a different file for materials, BOMS, and more.

2.2.1.1.2 Importieren von Projekten in Ihren Arbeitsbereich

Verwendung

Als Modeler können Sie mit mehreren Arbeitsbereichen arbeiten, von denen jeder einen eigenen, logisch getrennten Satz an Projekten aufweist. In einigen Fällen müssen Sie möglicherweise ein Projekt aus einem Arbeitsbereich in einen anderen kopieren oder möchten an einem Modell arbeiten, das von einem Kollegen angelegt wurde. Dies können Sie tun, indem Sie das Projekt Ihrem Arbeitsbereich hinzufügen.

Vorgehensweise

1. Klicken Sie in der Sicht *Projektexplorer* mit der rechten Maustaste auf den Leerraum.
2. Wählen Sie *Importieren*.
3. Öffnen Sie im Fenster *Importieren* den Ordner *Allgemein*, und wählen Sie *Vorhandene Projekte in den Arbeitsbereich*.
4. Wählen Sie *Weiter*.
5. Entscheiden Sie, ob Sie das Projekt aus der Ordnerstruktur in Ihrem Dateisystem oder aus einer Archivdatei importieren möchten.

Importieren aus dem Dateisystem

1. Wählen Sie das Optionsfeld *Stammverzeichnis auswählen*.
2. Wählen Sie über die Drucktaste *Durchsuchen...* das Stammverzeichnis mit den Projekten aus.
3. Wählen Sie das zu importierende Projekt aus.

   > i Note
   >
   > Ausgegraute Projekte können nicht ausgewählt werden, da Sie bereits in Ihrem Arbeitsbereich vorhanden sind.

4. Um eine Kopie dieses Projekts zu erstellen und sie in Ihren Arbeitsbereich zu kopieren, wählen Sie *Projekte in Arbeitsbereich kopieren*. Wenn Sie die Dateien im derzeitigen Speicherort bearbeiten möchten, wählen Sie nicht diese Option.
5. Wählen Sie *Beenden*.
   Das Projekt wird dann im *Projektexplorer* angezeigt.

Importieren aus einer Archivdatei

1. Wählen Sie das Optionsfeld *Archivdatei auswählen:*, und verwenden Sie die Drucktaste *Durchsuchen...*, um die Archivdatei auszuwählen, die das Projekt enthält.
2. Wählen Sie das zu importierende Projekt aus. Die Option *Projekte in Arbeitsbereich kopieren* ist standardmäßig ausgewählt und ist erforderlich.

   > i Note
   >
   > Ausgegraute Projekte können nicht ausgewählt werden, da Sie bereits in Ihrem Arbeitsbereich vorhanden sind.

3. Wählen Sie *Beenden*.
   Das Projekt wird dann im *Projektexplorer* angezeigt.

2.2.1.2 Hinzufügen von Referenzprojekten zu einem vorhandenen Projekt

Verwendung

Es ist üblich (sogar ein bewährtes Geschäftsverfahren), ein oder mehrere Projekte anzulegen, die allgemeine Routinen zur Wiederverwendung in anderen Modellprojekten enthalten. Um Inhalte von einem anderen

Projekt wiederzuverwenden, kennzeichnen Sie das Projekt als „Referenzprojekt“. Der Inhalt wird dann in der Lösungsmodellierungsumgebung so behandelt, als wäre er in einem einzelnen Projekt enthalten.

> i Note
>
> Wenn eines der folgenden Objekte:
>
> - Klasse
> - Material
> - Merkmal
> - Constraint
> - Regel
> - Variantentabelle
> - Constraintnetz
> - Regelnetz
> - PFunction
>
> in mehreren Projekten definiert wurde, wird es als Dublettenfehler gekennzeichnet. Referenzprojekte ermöglichen Ihnen die Wiederverwendung von Objekten, ohne sie mehr als einmal definieren zu müssen.

Vorgehensweise

Mit folgendem Verfahren können Sie Referenzprojekte anzeigen:

1. Öffnen Sie im Fenster *Importieren* den Ordner *Allgemein*, und wählen Sie *Dateisystem*.
   Klicken Sie auf *Weiter*.
2. Geben Sie mit der Drucktaste *Durchsuchen* die zu importierenden Ordner und Dateien an.
   Klicken Sie auf *Fertigstellen*.

2.2.1.3 Exportieren eines Modellprojekts

Kontext

Mit diesem Verfahren exportieren Sie ein Lösungsmodellprojekt, um es für Ihre Kollegen freizugeben.

Vorgehensweise

1. Klicken Sie in der Sicht *Projektexplorer* mit der rechten Maustaste auf den Projektordner.
2. Wählen Sie *Exportieren*.

3. Öffnen Sie im Fenster *Exportieren* den Ordner *Allgemein*, und wählen Sie *Archivdatei*.
4. Wählen Sie *Weiter*.
5. Wählen Sie die Drucktaste *Alles auswählen*, und geben Sie den Speicherort und den Namen der anzulegenden Archivdatei an.
6. Wählen Sie unter *Optionen* die Optionen für Dateiformat, Verzeichnisstruktur und Komprimierung.
7. Wählen Sie *Beenden*.

   Das Projekt wird dann in die von Ihnen angegebene Datei exportiert.

2.2.1.4 Anlegen einer Startkonfiguration

Kontext

Mit diesem Verfahren legen Sie eine Startkonfiguration aus der Java- oder Debug-Perspektive an.

Vorgehensweise

1. Wählen Sie *Perspektive öffnen* > *Java-Perspektive*.
2. Wählen Sie *Ausführen* > *Ausführungskonfigurationen...* oder *Ausführen* > *Debug-Konfigurationen...*.
3. Erweitern Sie den Knoten *Eclipse-Anwendung*, und geben Sie einen Namen für die Konfiguration ein.

   Klicken Sie auf *Anwenden* und dann auf *Ausführen*.

Ergebnisse

Die neue Konfiguration ist einsatzbereit.

2.2.1.5 Definieren benutzerdefinierter Funktionen

Verwendung

Mit diesem Verfahren legen Sie benutzerdefinierte Funktionen in der Eclipse-basierten Lösungsmodellierungsumgebung an.

Vorgehensweise

1. Wählen Sie *Datei* > *Neu* > *Leere Modelldatei*
   Es wird ein Dialogfenster angezeigt, in dem Sie dazu aufgefordert werden, einen Ordner auszuwählen und einen Dateinamen für die neue Funktion einzugeben.
2. Geben Sie die erforderlichen Daten ein.
3. Wählen Sie *Fertigstellen*.
   Im Texteditor wird eine leere Datei angezeigt.
4. Drücken Sie `STRG` + `LEERTASTE`, und doppelklicken Sie auf *function*.
5. Geben Sie die erforderlichen Daten ein, und sichern Sie die neue Funktion.

i Note

Die kompilierte Java-Klasse muss dem Klassenpfad der EJB-IPC-Pakete (`com.sap.custdev.projects.fbs.slc.ejb-ipc`) hinzugefügt werden. Dies kann über ein Fragment oder die Funktion `Eclipse-RegisterBuddy: com.sap.custdev.projects.fbs.slc.ejb-ipc` erfolgen.

Das neue Fragment oder Plug-In mit der Funktion kann an die Eclipse-Umgebung selbst übergeben werden (MyEclipse\plugins oder MyEclipse\dropins), oder Sie können die Test-Benutzungsoberfläche in Eclipse in einer Laufzeitkonfiguration starten, die alle erforderlichen Pakete enthält.

Weitere Informationen hierzu finden Sie in SAP-Hinweis 1701098 (Dokumentationsaktualisierung für Implementierung von Variantenfunktionen).

Beispiel

```
function FUNCTION {
name "Name"
characteristics
CSTIC1 primary,
CSTIC2,
}
```

Related Information

Deklarative Funktionen [Seite 58]
PFunction [Seite 60]

2.2.1.6 Exportieren von Projekten

Mit der Lösungsmodellierungsumgebung können Sie Wissensbasen in verschiedene Datenbanken wie auch eine Datei in das System exportieren.

Um einen erfolgreichen Export sicherzustellen, müssen Sie verschiedene Datenbankverbindungen und die erforderlichen Treiber für die Datenbank, in die Sie das Projekt exportieren, einrichten.

Bei Verbindungen zu lokalen Datenbanken werden folgende Faktoren berücksichtigt:

- MsSQL
- MySQL
- HANA
- Oracle
- MaxDB

Abgesehen von lokalen Datenbanken können Wissensbasen auch in die *CRM*- und *ECC*-Systeme exportiert werden.

2.2.1.6.1 Setup of Local Database Connection

Context

The SME can create connections to the following local databases:

- MsSQL
- MySQL
- HANA
- Oracle
- SAP MaxDB

You can setup a connection to the local database using the procedure below:

Procedure

1. Go to *Eclipse* > *Windows* > *Preferences* > *Expand SAP Solution Sales* > *Connections*.
2. Select the *Add* > *Name* .

   Select the relevant radio button.
3. Select the *Database Type*.
4. Enter the *Server Name* and *Database Name*.
5. Enter the port and client, if this information is not populated automatically.
6. Enter the login name for the database.
7. You can add the drivers using the procedure below:
   a. Go to *Eclipse* > *Windows* > *Preferences* > *Expand SAP Solution Sales*.

b. Add the driver pertaining to the database connection that is being added.

   Click on *OK*.

Example

You can further refer to the example below for more information:

1. Select Microsoft SQL Server as the *Database Type*.
2. Set localhost as the *Server*.
3. Enter MsSQL Database as the *Database Name*.
   This value is user defined.
4. The port and client should be auto-populated as **1433** and **000**, respectively.
5. Enter *sa* as the login name.
   This value is provided during the DB creation process.
6. Add *Microsoft SQL Server JDBC4 Driver* in the *Driver* section.
   Click on *OK*.

2.2.1.6.2 Setup of CRM Connection

Context

You can use the following procedure to export a project to CRM:

Procedure

1. Go to *Eclipse* > *Windows* > *Preferences* > *Expand SAP Solution Sales* > *Connections*.
2. Select the CRM radio button to add a name for the project.
3. The name will be user defined and relevant for the CRM connection.
4. Add the CRM *System Number* into the consideration for the connection.
5. Under the *Application Server*, enter the server details from the properties section of the CRM system.
6. Enter the *Client* number for the system.
7. Enter the login name defined above to connect to the CRM system.
8. You can refer to the example below for more information:
   a. *Application Name* = A valid name
   b. *System Number* = 11
   c. *Application Server* = Server details from the properties section

d. *Client* = 700

e. *Login Name* = UNIT_TEST

> **i Note**
>
> No drivers are required for this connection.

2.2.1.6.3 Setup of ERP Connection

Context

You can export a project to ERP using the following procedure:

Procedure

1. Go to *Eclipse* > *Windows* > *Preferences* > *Expand SAP Solution Sales* > *Connections*.
2. Select the ECC radio button to enter the relevant name.
3. Add a *Name* to the ECC connection.

   This name is user defined.
4. Add the *System Number* of the ECC system in the consideration for the connection.
5. Under the *Application Server*, enter the server details from the properties of the ECC system.
6. Enter the *Client* number of the system.
7. Enter the login name to connect to the ECC system.
8. You can refer to the example below for further information:
   a. *Application name* = A valid name
   b. *System number* = 11
   c. *Application server* = Server details from the properties section
   d. *Client* = 700
   e. *Login name* = UNIT_TEST

> **i Note**
>
> No drivers are required for this connection.

2.2.1.6.4 Exporting a Knowledge Base to Database

Use

You use this procedure to export a knowledge base and create a runtime version from it.

You can export a knowledge base from the *File* menu, from the *Project Explorer* view, or from the *Model Explorer* view.

Procedure

From the File Menu

1. Choose *File* › *Export* › *SAP Solution Sales Configuration* › *Export Knowledge Base*
2. Select the knowledge base you want to export
3. Choose *Next*
4. Select the connection for the target database
5. Optionally, you can also do the following:
    - Deactivate the validation option in the *Export Knowledge Bases* (multiple KB export) dialog

      > ⚠ Caution
      >
      > If you decide to deactivate the validation, the status of the data in the back end cannot be guaranteed after the export.

    - Activate the *Include local variant table content during export* option
6. Enter the password and choose *Finish*.

From the Project Explorer View

1. Right-click anywhere in the *Project Explorer* and choose *Export* › *SAP Solution Sales Configuration* › *Export Knowledge Base*.
2. Select the knowledge base you want to export.
3. Optional in the *Export Knowledge Bases* dialog (multiple kb export dialog): Deactivate the validation option.

   > ⚠ Caution
   >
   > If you decide to deactivate the validation, the status of the data in the back end cannot be guaranteed after the export.

4. Choose *Next*.
5. Select the connection for the target database.
6. Enter the password.
7. Optional: Activate the *Include local variant table content during export* option.
8. Choose *Finish*.

From the Model Explorer View

1. Right-click the knowledge base definition and choose *Export Knowledge Base*.

2. Select the connection for the target database.
3. Enter the password and choose *Finish*.

> ℹ Note
>
> Every time you export a knowledge base, the system asks you whether you want to start a test session in the testing perspective.

> ⚠ Caution
>
> If a validation error is detected in any of the dependent `.ssc` files when you export a knowledge base (for example, due to unrelated `class`, `cstic`, `material`, or `variant` tables present in that file, an error will be thrown that prevents you from exporting the knowledge base.

2.2.1.6.5 Exporting a Knowledge Base to a File

Follow this process to export a knowledge base from the workspace to a local folder.

Context

This procedure exports all the `.xml` files to the local folder of the knowledge base. You can export a knowledge base from the *File* menu either from the *Project Explorer* or the *Model Explorer* view.

Procedure

1. Setup the file connection.
   a. Go to *Eclipse* > *Windows* > *Preferences* > *Expand SAP Solution Sales* > *Connections*.
   b. Select *Add Name*.
   c. Select the relevant radio button for *File*.
   d. Provide the path to the selected local folder.
   e. Click on *OK*.
2. Export the knowledge base.
   - **Exporting from the file menu**
     1. Choose *File* > *Export* > *SAP Solution Sales Configuration* > *Export Knowledge Base*.
     2. Select the knowledge base you want to export.
     3. Optionally, you can also deactivate the option for validation in the *Export Knowledge Bases* dialog for multiple knowledge base export.
     4. Choose *Next*.
     5. Select the file connection created for the export.
     6. After the export is completed, the local folder should contain all the relevant `.xml` files of the project.

- **Exporting the project explorer view**
    1. Right click anywhere in the project explorer and choose *Export* > *SAP Solution Sales Configuration* > *Export Knowledge Base*.
    2. Select the knowledge base you want to export.
    3. Optionally, you can also deactivate the option for validation in the *Export Knowledge Bases* dialog for multiple knowledge base export.
    4. Choose *Next*.
    5. Select the file connection created for export.
    6. After the export is completed, the local folder should contain all the relevant `.xml` files of the project.
- **Exporting form the model explorer view**
    1. Right click on the knowledge base definition and choose *Export knowledge Base*.
    2. Select the file connection created for the export.
    3. After the export is completed, the local folder should contain all the relevant `.xml` files of the project.

2.2.1.7 KB Admin Tool Support

2.2.1.7.1 Uploading Knowledge Bases

You use this procedure to upload a knowledge base into the local database. On the modeling user interface (UI), you can upload the knowledge base using the *Upload Knowledge Base* menu. You can upload the knowledge base as xml files.

Procedure

1. Go to *SAP Modeling* > *Upload Knowledge Base*.
2. Choose the directory containing XML files.
3. Choose *Next*.
4. Choose the connection and enter the required details.
5. Choose *Finish*.

More Information

While downloading a runtime version via `CU36` utility, some text formatting may be lost and replaced by hashes.

For more information about handling this issue, refer to SAP Note 2339258 (Japanese and Chinese texts are lost in runtime version.).

2.2.1.7.2 Wissensbasen löschen

Über die Option *Wissensbasis löschen* in der Lösungsmodellierungsumgebung können Sie Wissensbasen aus unterstützten Datenbanken löschen.

Vorgehensweise

1. Wählen Sie *SAP-Modellierung* > *Wissensbasis löschen*.
2. Wählen Sie die relevanten Verbindungsdetails aus, und klicken Sie auf *Weiter*.
3. Wählen Sie die Wissensbasen aus, die Sie löschen möchten, und wählen Sie *Fertigstellen*.
4. Wählen Sie *OK*, um fortzufahren.

Ergebnisse

Die ausgewählten Wissensbasen werden aus der Datenbank gelöscht, und es wird eine Bestätigungsmeldung angezeigt.

2.2.1.7.3 Uploading External Variant Tables

Context

Normally, external variant tables are replicated from the backend system to local DB via SME dataloader.You may, however, want to edit and test the data frequently in the SME before maintaing new data in the backend.You use this procedure to upload external variant tables to local databases that can be downloaded from the ECC and CRM systems.

i Note

External variant tables should be created in the backend and then downlaoded to an empty folder on your PC via transaction `/n/SLCE/CFG_SUPPORT`. This creates two files in this folder. *meta* file contains the table header information and actual data is contained in *data* file. Only *data* file should be updated with new data values for upload in the SME.

Procedure

1. Choose *File* > *Upload External Variant Tables*.
2. Browse the directory where the external variant table was downloaded previously.

3. Choose *Next*.
4. Enter the connection details of the database to which the variant tables are to be uploaded.
5. Choose *Finish*.

2.2.2 SAP-Testing-Perspektive

Die *SAP-Testing*-Perspektive wird automatisch geöffnet, wenn Sie eine Wissensbasis öffnen. Diese Perspektive hat ihre eigenen zugehörigen Sichten und Editoren. Dazu gehören Konfigurationseditor, Merkmalssicht, Nichtteil-Instanzensicht, Eigenschaften-Sicht und eine Reihe von Debug-/Analyse-Sichten.

Komponente	Beschreibung
Konfigurationseditor	Zeigt den Inhalt der Konfigurationssitzung in einer Baumstruktur an. Der Knoten der höchsten Ebene zeigt Datenbanknamen, Wissensbasisnamen, Profil und Version der Wissensbasis an, die zum Starten dieser Konfigurationssitzung geöffnet wurde. Direkt unter dem Wissensbasisknoten befindet sich der Konfigurationsknoten, der den Namen der Konfiguration anzeigt (identisch mit dem Wissensbasisnamen). **Note** Durch die Wissensbasiskoordination können mehrere, durch ihre eigenen Wissensbasen gesteuerte Konfigurationen gleichzeitig aktiv sein. Diese werden "koordiniert", um als eine Konfiguration zusammenzuarbeiten. Unterhalb des Konfigurationsknotens sind die Klassen- und Materialinstanzen, unter denen sich die Merkmale mit ihren zugeordneten Werten befinden. Ein grünes Rechteck gibt an, dass alle erforderlichen Werte der Instanz zugeordnet wurden, diese also vollständig ist. Ein gelbes Dreieck gibt an, dass die Instanz unvollständig ist, d.h., einem obligatorischen Merkmal wurde kein Wert zugeordnet. Ein roter Kreis gibt einen Konflikt oder eine inkonsistente Konfiguration an. Für Unterstützung beim Debugging eines Konflikts öffnen Sie die Sicht *Konflikte*, und klicken Sie auf eine beliebige Position, die in der Konfiguration mit einem roten Kreis gekennzeichnet ist. Informationen zum Konflikt werden in der Sicht "Konflikte" dargestellt. Sie können den Konfigurationsinhalt in diesem Editor bearbeiten, indem Sie mit der rechten Maustaste auf eine Position oder einen Knoten klicken. Alle zulässigen Aktionen sind dann zur Auswahl verfügbar. Um der Konfiguration eine Instanz hinzuzufügen, verwenden Sie die Sicht *Nichtteil-Instanz*.

Komponente	Beschreibung
Merkmale-Sicht	In der Sicht "Merkmale" werden die sichtbaren Merkmale der ausgewählten Instanz im Konfigurationseditor aufgeführt. Werte können einem beliebigen Merkmal zugeordnet werden, es sei denn, es wurde im Modell als "keine Eingabe" oder als ein Wert gekennzeichnet, der bereits von einem Constraint festgelegt wurde. Obligatorische Merkmale werden mit einem gelben Dreieck gekennzeichnet, auch wenn ihnen ein Wert zugeordnet wurde. Klicken Sie auf das umgedrehte Dreieck im Kopf der Sicht, um das Menü anzuzeigen. Sie können das Menü zum Ein-/Ausblenden unsichtbarer Merkmale und zum Anzeigen sprachabhängiger und technischer Namen verwenden.
Nichtteil-Instanzen-Sicht	Mit der Sicht "Nichtteil-Instanzen" werden Instanzen von Klassen oder Materialien manuell der Konfiguration hinzugefügt. Wenn Sie auf eine beliebige Position in der Sicht "Nichtteil-Instanzen" doppelklicken, wird sie als eine Instanz dieser Art in Form einer "freistehenden", unabhängigen Instanz hinzugefügt. Sie hat keine "Teil-von"-Relation zur Stamminstanz, sofern nicht ein Constraint dem Modell hinzugefügt wurde, um eine Beziehung herzustellen. Sie wird als "Nichtteil-Instanz" bezeichnet, weil sie kein Bestandteil von einem anderen Element ist. **i Note** Sie können Komponenten bei einer bestimmten Stücklistenposition einem Material hinzufügen, indem Sie im Konfigurationseditor mit der rechten Maustaste auf das Material klicken. Wenn eine Stückliste für das Material definiert wurde, steht die Option *Komponente hinzufügen* zur Auswahl. Wenn sie ausgewählt wurde, wird eine "Teilinstanz" als eine Komponente dieses Materials hinzugefügt. Nichtteil-Instanzen können nicht auf diese Weise hinzugefügt werden. In diesem Fall müssen sie über die Sicht "Nichtteil-Instanzen" hinzugefügt werden.
Eigenschaften-Sicht	In der Sicht "Eigenschaften" werden Informationen zu allen Eigenschaften der aktuellen Position (der ausgewählten Position in der aktiven Sicht) angezeigt. Dies kann eine Instanz im Konfigurationseditor oder in der Sicht "Merkmale" oder ein Constraint in der Sicht "Rechtfertigungen" sein. Jede Position, die Sie in einer Sicht auswählen, hat Eigenschaften, die in dieser Sicht angezeigt werden.

Debug- und Analyse-Sichten

Diese Sichten werden als ein Set über den unteren Bereich der SAP-Testing-Perspektive dargestellt.

Komponente	Beschreibung
Test-Runner	Erfasst alle Aktionen und zeichnet sie in einer Skriptdatei auf, die gesichert und erneut ausgeführt werden kann. Über die Drucktaste "Ausführen" wird ein importiertes Skript komplett ausgeführt. Um nur einen bestimmten Schritt auszuführen, wählen Sie die letzte auszuführende Zeile aus, klicken Sie mit der rechten Maustaste und wählen *Ausführen bis Zeile*. Mit der Drucktaste "Exportieren" wird der derzeitige Test-Runner-Inhalt in eine Datei exportiert. Sie werden dazu aufgefordert, den Verzeichnis- und Dateinamen einzugeben. Mit der Drucktaste "Importieren" wird ein Testskript importiert. Sie werden dazu aufgefordert, den Verzeichnis- und Dateinamen einzugeben. Die Datei muss das Suffix `.performer` aufweisen. Mit der Drucktaste "Zurücksetzen" wird die Konfigurationssitzung in ihren ursprünglichen Zustand zurückgesetzt. Um dem Skript erwartete Ergebnisse hinzuzufügen, wählen Sie ein Objekt im Konfigurationseditor aus, klicken Sie mit der rechten Maustaste und wählen eine der Optionen *... erwartet*. Bei der Wiedergabe eines Skripts, das erwarteten Inhalt umfasst, wird der aktuelle Inhalt der Konfiguration in der Lösungsmodellierungsumgebung mit dem aufgezeichneten erwarteten Inhalt verglichen. Alle Übereinstimmungen werden grün markiert, etwaiger unerwarteter Inhalt wird rot markiert, und es wird eine Erklärung des Deltas bereitgestellt.
Konflikte-Sicht	Wenn zwei gegensätzliche Fakten erkannt werden, wird von der Engine ein Konflikt ausgelöst. In der Sicht "Konflikte" werden die mit der Instanz, die aktuell im Konfigurationseditor ausgewählt ist, verknüpften Konflikte aufgeführt. Außerdem werden Ratschläge zur Behebung des Konflikt in Form von "Konfliktannahmen" erteilt.

Komponente	Beschreibung
Profiling-Sicht	Profiling ist eine Funktion der Configuration Engine, mit der die Ausführung von Beziehungen verfolgt und die Anzahl der Ausführungen für jede einzelne Beziehung und die gesamte verstrichene Zeit erfasst wird. Das Profiling muss aktiviert werden, um das Aufzeichnen von Statistiken zu starten; wenn es deaktiviert wird, werden die Ergebnisse angezeigt. Um das Profiling zu starten oder zu stoppen, klicken Sie in der Symbolleiste der Sicht "Profiling" auf den Pfeil nach unten und wählen *Profiling starten/stoppen*.
Rechtfertigungen-Sicht	Jeder Instanz- und Merkmalswert wird durch eine Reihe von Fakten gerechtfertigt, die durch den Benutzer oder durch Beziehungen bestimmt werden, und wird von der Rechtfertigungsverwaltung erfasst. In der Sicht "Rechtfertigungen" werden alle Fakten und die Beziehungen, die sie bestimmten, angezeigt. Zusammen rechtfertigen diese Fakten und Beziehungen die Position, die aktuell ausgewählt ist.
Trace-Sicht	In der Sicht "Trace" werden alle Engine-Aktivitäten ausführlich aufgeführt. Weitere Informationen erhalten Sie unter Tracing [Seite 98].

2.2.2.1 Lokales Testen von Modellen

Kontext

Mit diesem Verfahren testen Sie Ihr Modell, bevor Sie es zur Verwendung im Kundenauftragsprozess in das Zielsystem exportieren. In der Test-Benutzungsoberfläche und Test-Configuration-Engine können Sie die Lösung konfigurieren, um sicherzustellen, dass das Modell wie erwartet ausgeführt wird. Sie können die Lösung manuell oder über ein erfasstes Test-Skript konfigurieren. Darüber hinaus können Sie die Leistung von Constraints und Beziehungen messen.

Vorgehensweise

1. Wählen Sie *Datei* > *Wissensbasis öffnen...*
2. Wählen Sie die *Verbindung* aus, und geben Sie die erforderlichen Details ein.

3. Wählen Sie *Weiter*.
4. Wählen Sie eine Wissensbasis und ein Profil.
5. Wählen Sie *Weiter*.

Ergebnisse

Die Test-Benutzungsoberfläche wird geöffnet. Während Sie die Lösung in der Test-Benutzungsoberfläche konfigurieren, werden Informationen auf den folgenden Registerkarten angezeigt:

- **Test-Runner**
  Alle von Ihnen in der Test-Benutzungsoberfläche durchgeführten Aktionen werden vom System erfasst, und das resultierende Skript wird auf der Registerkarte *Erfasstes Skript* angezeigt. Das Skript kann in einer Datei gesichert werden. Sie können ein gesichertes Skript auf der Registerkarte *Ausführbares Skript* öffnen und dann ausführen. Sie können das Skript komplett oder nur bis zu einer bestimmten Zeile ausführen. Wenn Sie ein gesichertes Skript ausführen und dann mit der Konfiguration der Lösung fortfahren, fügt das System Ihre Aktionen dem Skript hinzu.
- **Konflikte**
  Die Modellkonflikte, die beim Testen des Lösungsmodells aufgetreten sind, werden im System angezeigt.
- **Profiling**
  Es werden alle während der Konfigurationssitzung ausgeführten Constraints angezeigt. Für jeden Constraint wird die Anzahl an Aufrufen und die Ausführungszeit in Millisekunden angegeben. Sie können einen Spaltenkopf auswählen, um die Daten nach dieser Spalte zu sortieren.
- **Rechtfertigungen**
  Es werden die Rechtfertigungen für Konfigurationsregeln angezeigt.
- **Trace**
  Die während der Konfigurationssitzung durchgeführten Schritte werden angezeigt.
- **Nichtteil-Instanzen**
  Die Sicht zeigt alle Nichtteil-Instanztypen an. Sie können eine Nichtteil-Instanz anlegen, indem Sie auf einen Typ doppelklicken und ihn dann zum Testen zu Ihrer Konfiguration hinzufügen.

> i Note
>
> Nachdem Sie Ihr Modell lokal getestet haben, können Sie die Wissensbasis in den Verkaufsvorgängen im SAP-CRM- oder SAP-ECC-Backend-System testen. Hierzu müssen Sie die Wissensbasis so in eine CRM- oder ECC-Datenbank exportieren wie Sie sie in die lokale Datenbank exportieren. In diesem Fall wählen Sie jedoch statt der lokalen Datenbankverbindung die CRM- oder ECC-Verbindung. Weitere Informationen finden Sie unter Exportieren von Wissensbasis-Laufzeitversionen [Seite 86].

2.2.2.1.1 Test-Runner

In der Sicht „Test-Runner“ können Sie die Testskripte teilweise oder vollständig ausführen. Darüber hinaus sind in der Sicht Aktionen zum Speichern und Laden von Testskripten verfügbar. Beim Ausführen eines Testskripts wird die Konfigurationsstruktur den Aktionen im Testskript entsprechend geändert. Im Fall von Erwartungen wird die Konfigurationsstruktur geprüft. Wenn die Erwartung nicht dem Istzustand der Konfigurationsstruktur entspricht, wird die Abweichung in der Sicht „Test-Runner“ gemeldet. Die geladenen Performer-Skripte werden auf der Registerkarte *Geladene Dateien* angezeigt.

Über das Testskript (Performer-Testskript) werden sämtliche Benutzerbefehle als Testschritte aufgezeichnet, damit sie gespeichert und zu einem späteren Zeitpunkt ausgeführt werden können. In der Lösungsmodellierungsumgebung können „Erwartungen“ aufgezeichnet werden. Dies bietet Ihnen die Möglichkeit, ein Modell nach der Ausführung von Benutzerbefehlen zu prüfen und sicherzustellen, dass die Erwartungen – beispielsweise nach einer Modelländerung – weiterhin zutreffen. Die Testskripte und die Test-Runner-Benutzungsoberfläche dienen somit nun als einfaches Werkzeug für automatisierte Modelltests.

2.2.2.1.1.1 Saving Test Scripts (Performer)

You use this procedure to save an SAP Solution Sales Configuration test script in the Eclipse-based Solution Modeling Environment.

Procedure

1. Open the knowledge base.
2. Modify the model configuration.
3. Choose *Save Script* in the Test Runner view.
4. Enter the file name and save.

   The test script is saved in the specified location on the file system in the `.performer` format.

   It contains the action performed by the user on a configuration and also records the expectation when *Expect All on Save* is selected while saving the performer file. These performer scripts are also enhanced to contain the configuration xml in a zipped format. It contains various flags for restoration and performance checks which can we used when running the performer in the headless mode.

   2689606 - SME Performer and Restore Configuration Feature Enhancements

   2777356 - API to Apply Engine Settings

Next Steps

After you have saved the test script, you can load, run, and reset the script.

2.2.2.1.1.2 Loading Test Scripts (Performer)

You use this procedure to load an SAP Solution Sales Configuration test script (performer files) in the Eclipse-based Solution Modeling Environment.

Context

You can use this script to test the configuration for the model and can load a single or multiple test scripts to set the configuration.

Procedure

1. Choose *Load Script* in the *Test Runner* view.
2. Select the test-script files and choose *Open*.

   Note

   If multiple performer scripts are loaded, they must be of the same KB RTV.

Results

The performer test scripts are displayed on the *Loaded Files* tab. The loaded performer script would only be available in the performer run session, and not in the other configuration view launched for restore when performer is run with flags enabled for restore.

2.2.2.1.1.3 Running Test Scripts (Performer)

You use this procedure to run an SAP Solution Sales Configuration test script in the Eclipse-based Solution Modeling Environment.

Procedure

1. Choose *Run Script* in the *Test Runner* view.
2. Double-click on the test script on the *Loaded Files* tab for details.

   The *Executable Script* tab opens with the test script details.

Results

The performer test scripts run with the status *SUCCESS* or *FAILURE*.

2.2.2.1.1.4 Resetting Test Scripts (Performer)

You use this procedure to reset an SAP Solution Sales Configuration test script in the Eclipse-based Solution Modeling Environment.

Context

The target configuration is set when a test script for a model is executed. You can choose to reset either one or multiple configuration scripts, as they are loaded in the test runner.

Procedure

1. Open the *Test Runner* view.
2. Choose either *Reset Configuration* (to reset a single configuration script) or *Reset All Configurations* (to reset multiple configuration scripts).

Results

The performer test script is reset to *Not Run*.

2.2.2.1.1.5 Deleting Test Scripts (Performer)

You use this procedure to delete SAP Solution Sales Configuration test scripts in the Eclipse-based Solution Modeling Environment.

Procedure

1. Select the test script performer on the *Loaded Files* tab.
2. Choose *Delete File* in the *Test Runner* view.

Results

The performer test script is deleted from the *Loaded Files* tab.

2.2.2.1.2 Problemfilter auf Registerkarte „Ausführbares Skript“

Mit diesem Verfahren filtern Sie Aktionen in der Eclipse-basierten Lösungsmodellierungsumgebung. Sie können die Aktionen anhand der Problemerläuterung filtern und so mühelos die bei der Testskriptausführung erkannten fehlerhaften Aktionen ermitteln.

Vorgehensweise

1. Öffnen Sie die Registerkarte *Ausführbares Skript*.
2. Geben Sie im *Problemfilter* einen Text basierend auf der Erläuterung des Problems ein.

> **i Note**
>
> Mehrere geladene Performer-Skripte müssen dieselbe Wissensbasis-Laufzeitversion haben.

2.2.2.2 Tracing and Logging

Use

The tracing function enables you to record selected engine activities.

Prerequisites

- You have activated the trace by choosing the *View Menu* pushbutton in the *Trace* view and then choosing *Configure*. In the trace configuration window, choose *Enable Tracing*.

> **i Note**
>
> For performance improvements in SME, traces would be updated only when the user clicks on the *Refresh* button.

- You have selected the types of activity to be traced.

Types of Traceable Activity

Type	Description	Comments
DDB (Dynamic database)	The DDB is the engine-internal storage for facts maintained by the inference engine. Trace output of this type refers to changes in the DDB.	Tracks all assertion and retraction of facts in the dynamic database
PMS (Pattern matching system)	The PMS is responsible for matching patterns in a dependency against facts in the DDB. Dependencies are evaluated against the provided matches. In other words, the PMS is responsible for identifying the dependencies to be executed and for providing them with the objects (such as instances and characteristics) that are required for the dependency evaluation. Trace output of this type refers to activities inside the PMS.	Tracks all patterns in the configuration that are detected by the pattern matching system
CSTR (Constraint)	Trace output of this type refers to constraint evaluations.	Records the execution of any constraint
RULE	Trace output of this type refers to rule evaluations.	Records the execution of any rule
SCND (Selection condition)	Trace output of this type refers to the evaluation of selection conditions.	Traces the execution of any selection condition
PCND (Precondition)	Trace output of this type refers to the evaluation of preconditions.	Traces the execution of any preconditions
PROC (Procedure)	Trace output of this type refers to the evaluation of procedures.	Traces the execution of any procedures
FUNC (User-defined functions)	Trace output of this type refers to the evaluation of user-defined functions.	Records the execution of user-defined functions
TABL (Variant tables)	Trace output of this type refers to the access of variant tables.	Tracks interaction with a variant table

The trace result can be displayed in the configuration dialog. Each row is structured as follows:

```
[EngineTrace]:[ <Instance Number>- <Knowledge Base Name>] <Sequential Number for
Each Trace Row> <Related Engine Activity Type> <Message Number of the Text
Displayed for the Engine Activity Performed> <Engine Activity Performed>
```

Note

The message number of the text displayed for the engine activity performed corresponds to the numbers in ABAP message class 34.

Example

```
[EngineTrace]:[1-HSS_SOLUTION_KB] 1 DDB 204 New fact "sf($1, SCM_SALES_USERS,
VAL, 150.0)" inserted by "User"
```

Many expressions relating to the engine activity performed are self-explanatory. However, expressions about facts need further explanation.

A fact represents an elementary assertion about the state of the configuration. For example, if the user sets a characteristic `COLOR` to the value `BLUE`, a specific fact is created in the engine. The following types of facts can occur in the trace:

Fact Printed in Trace (Examples)	Explanation
`sf($1, COLOR, VAL, BLUE)`	This is a simple fact (sf). For the instance `$1`, this fact sets the value of the characteristic `COLOR` to `BLUE`. `VAL` refers to a specific facet of a characteristic. The available facets are described below.
`to($1, product: SAP_SYS)`	This is a type-of (to) fact which is used for classification. The instance `$1` is of the product type `SAP_SYS`
`rd($1, COLOR, BLUE, RED)`	This fact is about a restrictable domain (rd). For the instance `$1`, this fact sets the domain of values for the characteristic `COLOR` to the values `BLUE` and `RED`.

Facets of Characteristics

The tracing output of a simple fact (sf) always describes a certain facet of a characteristic. A facet is a property of a characteristic. The following facets exist:

- `VAL` (value facet): Value of a characteristic
- `DOM` (domain facet): Domain of a characteristic
- `REQ` (required facet): Specifies whether a characteristic is required
- `INV_P` (invisible facet): Specifies the visibility of a characteristic
- `NOINP_P` (no input facet): Specifies whether a characteristic is read-only

2.2.2.3 Import/Export Configuration

You can test your configuration on the test UI of the Solution Modeling Environment. In this way, you can mimic the restore scenario that is available on the user interface of SAP Solution Sales Configuration.

This feature in the Solution Modeling Environment enables you to perform the following tasks:

- Export configuration from the SME to XML
- Import configuration XML that has been exported from the SME
- Import configuration XML that has been exported from the user interface of SAP Solution Sales Configuration

- After the import, edit the configuration and export the updated configuration to XML

Export Configuration

On the *SAP Testing* user interface, you can export knowledge base configuration as a `*.cfg` file.

1. Open the knowledge base and complete configuration on the *SAP Testing* UI.
2. Right-click the tree view and go to *Configuration* > *Export*.
3. Enter a file name for your configuration.

Restore Configuration

In the Solution Modeling Environment, you can restore the configuration XML as follows:

1. Go to *File* > *Restore Configuration*.

   > **i Note**
   >
   > You must create a connection to the local database if a connection doesn't already exist.
2. Enter the database password and click *Next* to open the wizard.
3. Click *Browse* to select a configuration file. You can export files that have the formats `*.cfg` or `*.xml`.
4. Select the KB reference date and click *Finish* to see the restored configuration on the *SAP Testing* UI.

> **i Note**
>
> After the restore, you can edit the configuration on the *SAP Testing* UI. However, you cannot save a performer script for this updated configuration. (The *Save Script* button has been disabled in *Test Runner*).

2.2.2.4 DDB-, PMS- und TMS-Dumps

Die Lösungsmodellierungsumgebung stellt Dumps bereit, um den Benutzer bei der Analyse von Modellierungsproblemen zu unterstützen.

Folgende Dumps werden von der Lösungsmodellierungsumgebung bereitgestellt, um den Benutzern bei der Analyse von Modellierungsproblemen zu helfen:

- DDB-Dump (Dynamische Datenbank)
  Die DDB ist der interne Speicher für alle Fakten, die von der Herleitungs-Engine gepflegt werden. Die Trace-Ausgabe dieses Dumps bezieht sich auf Änderungen in der DDB.
- PMS-Dump (Pattern-Matching-System)
  Das PMS ist zum Abgleich von Mustern in einer Beziehung gegenüber Fakten in der DDB zuständig. Diese Beziehungen werden dann gegenüber den bereitgestellten Übereinstimmungen bewertet. Das PMS ist also dafür zuständig, die Beziehungen zu identifizieren, die ausgeführt werden sollen, und ihnen die Objekte (wie Instanzen und Merkmale) bereitzustellen, die für die Beziehungsauswertung erforderlich sind.

- TMS-Dump (Rechtfertigungsverwaltung)
  Das TMS ermittelt den Status von Fakten anhand der bekannten Mengen von Rechtfertigungen. Jegliche Statusänderungen werden an den TMS-Mandanten kommuniziert. Wenn eine neue Rechtfertigung an das TMS kommuniziert wird, wird der Status aller Fakten und Auslöser (inkrementell) aktualisiert. Der TMS-Dump führt außerdem bei Bedarf eine atms-Label-Berechnung für alle Fakten und Auslöser durch.

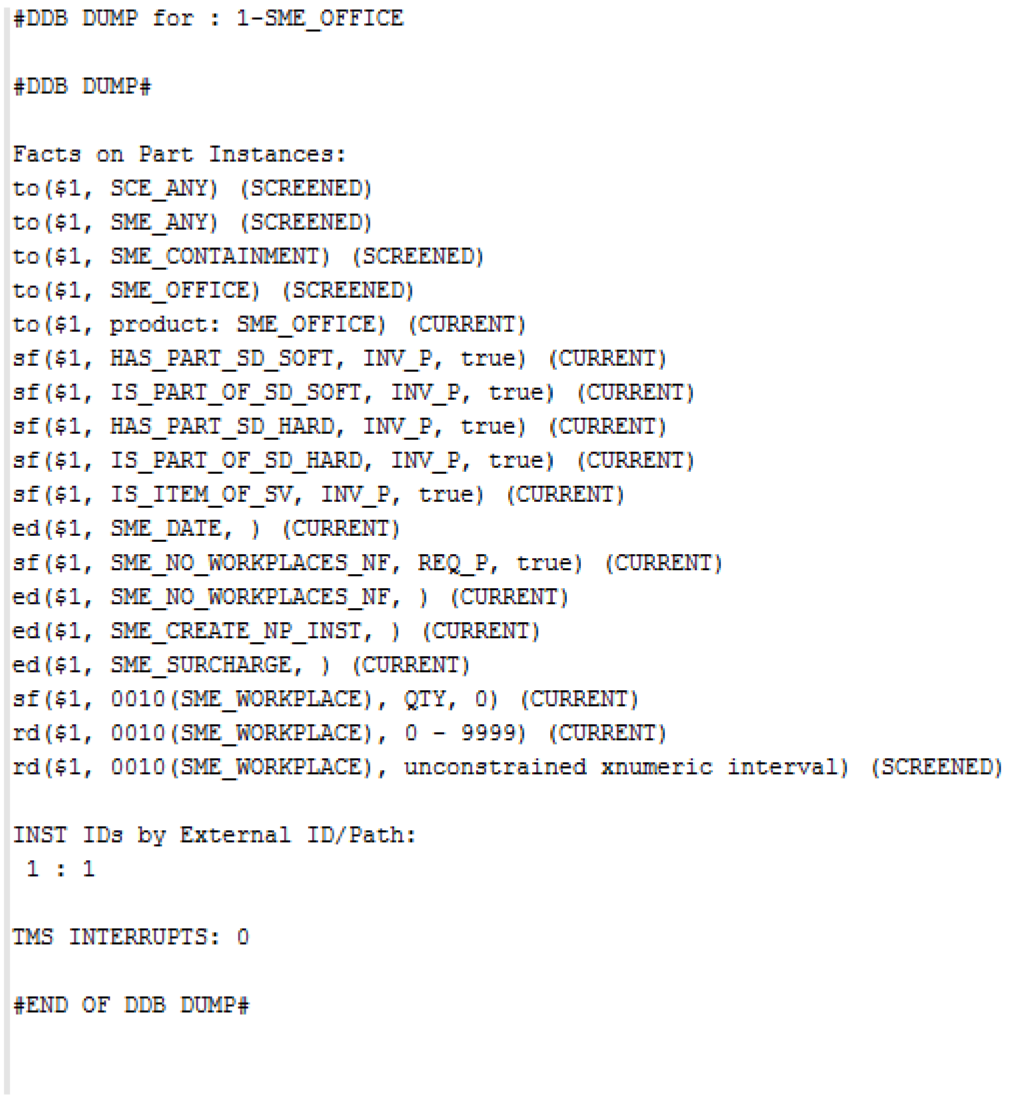
2.2.2.4.1 DDB-Dumps exportieren

Mit diesem Verfahren exportieren Sie einen DDB-Dump aus der Eclipse-basierten Lösungsmodellierungsumgebung.

Vorgehensweise

1. Klicken Sie in der Sicht *Modell* mit der rechten Maustaste auf die Wissensbasis, und wählen Sie *Konfiguration* > *DDB-Dump*.
2. Wählen Sie den gewünschten Speicherort für die Datei aus, und geben Sie einen Dateinamen ein.

Der DDB-Dump wird wie im Folgenden dargestellt exportiert:



```
#DDB DUMP for : 1-SME_OFFICE

#DDB DUMP#

Facts on Part Instances:
to($1, SCE_ANY) (SCREENED)
to($1, SME_ANY) (SCREENED)
to($1, SME_CONTAINMENT) (SCREENED)
to($1, SME_OFFICE) (SCREENED)
to($1, product: SME_OFFICE) (CURRENT)
sf($1, HAS_PART_SD_SOFT, INV_P, true) (CURRENT)
sf($1, IS_PART_OF_SD_SOFT, INV_P, true) (CURRENT)
sf($1, HAS_PART_SD_HARD, INV_P, true) (CURRENT)
sf($1, IS_PART_OF_SD_HARD, INV_P, true) (CURRENT)
sf($1, IS_ITEM_OF_SV, INV_P, true) (CURRENT)
ed($1, SME_DATE, ) (CURRENT)
sf($1, SME_NO_WORKPLACES_NF, REQ_P, true) (CURRENT)
ed($1, SME_NO_WORKPLACES_NF, ) (CURRENT)
ed($1, SME_CREATE_NP_INST, ) (CURRENT)
ed($1, SME_SURCHARGE, ) (CURRENT)
sf($1, 0010(SME_WORKPLACE), QTY, 0) (CURRENT)
rd($1, 0010(SME_WORKPLACE), 0 - 9999) (CURRENT)
rd($1, 0010(SME_WORKPLACE), unconstrained xnumeric interval) (SCREENED)

INST IDs by External ID/Path:
 1 : 1

TMS INTERRUPTS: 0

#END OF DDB DUMP#
```

DDB-Dump

2.2.2.4.2 PMS-Dumps exportieren

Mit diesem Verfahren exportieren Sie einen PMS-Dump aus der Eclipse-basierten Lösungsmodellierungsumgebung.

Vorgehensweise

1. Klicken Sie in der Modellsicht mit der rechten Maustaste auf die Wissensbasis, und wählen Sie *Konfiguration* > *PMS-Dump*.

2. Wählen Sie einen Speicherort für die Datei aus, und geben Sie einen Dateinamen ein.

2.2.2.4.3 TMS-Dumps exportieren

Mit diesem Verfahren exportieren Sie einen TMS-Dump aus der Eclipse-basierten Lösungsmodellierungsumgebung.

Vorgehensweise

1. Klicken Sie in der Modellsicht mit der rechten Maustaste auf die Wissensbasis, und wählen Sie *Konfiguration* › *TMS-Dump*.
2. Wählen Sie einen Speicherort für die Datei aus, und geben Sie einen Dateinamen ein.

2.2.2.5 Importing a Model in Local Database

Use this process to import a knowledge base from an SAP ECC or SAP CRM system using the data loader.

Use

You use this procedure to import the interface characteristics from compatible-mode knowledge bases in the source SAP ERP Core Component (SAP ECC) or Customer Relationship Management (CRM) system to the solution modeling environment database. You perform these steps in the Eclipse-based solution modeling environment.

Prerequisites

You have setup the data loader connection in the Solution Model Environment.

Procedure

1. Choose *File* › *Import...*.
   The system displays a dialog box prompting you to choose an import source.
2. Choose *SAP Solution Sales Configuration* › *Data Loader*.
3. Choose *Next*.

The system displays a dialog box prompting you to choose a Data Loader configuration from a list of the configured connections.

4. Select the configuration for the system and choose *Next*.
   The system can either be the `SAP ECC` system or the `CRM` system.
5. Enter the required passwords and choose *Finish*.

> i Note
>
> Compatible mode knowledge bases imported from the SAP ECC system can be tested using the test user interface; however they cannot be changed in the solution modeling environment.

For more information, see **Data Loader in the Solution Modeling Environment**.

More Information

Related Information

Datenlader in der Lösungsmodellierungsumgebung [page 109]

2.2.2.6 Importing a Model into a File

Prerequisites

A java project needs to be created, under which the relevant `.ssc` file is created.

Context

This procedure is used to import a knowledge base present in a backend database (CRM,ECC, local database) into a file, in an existing java project. The imported file contains the following details:

- Material definition
- Class definition
- Characteristic definition
- Bill of materials definition

as per the selected KB.

Procedure

1. In the solution modeling environment, choose ▶ *File Import* ▶ *SAP Solution Sales Configuration* ▶ *Import Model to File* ◀.

   Choose *Next*.
2. Choose one of the existing connection types: *CRM*, *SAP ECC*, or the local database.
3. Enter the *Client*, *User*, and *Password*.

   Choose *Next*.
4. Select the knowledge base runtime version for the product .

   You can sort the results by clicking on any one of the column headers.
5. Select the master data definitions you want to import.

   Choose *Include Dependent Bill of Materials* if you want to import the material definitions with the `boms` parameter. If you do not choose this option, the material definition will not contain a reference to a BOM.

   If you choose *Include Dependent Material Classifications*, the imported material definitions will include the `classes` parameter. If you do not choose this option, the material definition will not contain a reference to classes.

   > i Note
   >
   > These options affect only the material definition and not the import of bill of material or class definitions

   Choose *Next*.
6. Specify the folder and file names to be created, to store the definitions of the imported items.

   > i Note
   >
   > Do not select any advanced features.

   Choose *Finish*.

2.2.2.7 Importing a Model into a New Project

Context

This procedure is used to import a knowledge base present in a backend database (CRM,ECC, local database) into a Java project. The project then created will have individual folders of bills-of-materials, classes, characteristic and materials. Each folder shall have the relevant material definition, class definition, characteristic definition and bill of material definition in an .ssc file as per the elements selected during creation

Procedure

1. In the solution modeling environment, choose *File* > *Import SAP Solution Sales Configuration* > *Import Model to New Project*.

   Choose*Next*.
2. Choose one of the available connections: *ECC*, *CRM*, or the local database.
3. Enter the *Client*, *User*, and *Password*.

   Choose *Next*.
4. Select the knowledge base runtime version for the product.

   You can sort the results by clicking on any one of the column headers.
5. Select the master definitions you want to import.
6. Choose *Include Bill of Material* if you want to import the material definitions with the `boms` parameter. If you do not choose this option, the material definition will not contain a reference to a BOM.

   If you choose to *Include Dependant Material Classifications*, the imported material definitions will include the *classes* parameter. If you do not choose this option, the material definition will not contain a reference to classes.

   > i Note
   >
   > These options affect only the material definition and not the import of bill of materials definitions or class definitions.

   Choose *Next*.
7. Enter a project name under which the project folders will be created. The `.ssc` files with the selected parameter definitions will be saved here.

   Choose *Finish*.

2.2.2.8 Importing Master Data from SAP CRM/ECC

Use

At runtime in the SAP Solution Sales Configuration engine, the solution knowledge bases created in the solution modeling environment interact with the product knowledge bases created in SAP CRM/ECC.

Most of the master data for the referenced products is already defined in the SAP CRM/ECC system and can be reused in the solution modeling environment. These master data definitions are included in the knowledge base run time versions imported into the local SQL server database.

You use this procedure to import this master data to either a file or a new project.

Prerequisites

- You have configured a connection to your SAP ECC system in the Eclipse-based solution modeling environment.

Procedure

Importing Master Data to a File

1. In the solution modeling environment, choose *File* > *Import* > *SAP Solution Sales Configuration* > *Import Model to File*.
2. Choose *Next*.
3. Choose from an ECC or a CRM connection.
4. Enter the client, user, and password.
5. Choose *Next*.
6. Select the knowledge base runtime version for the product (you can sort the results by clicking any column header).
7. Select the master data definitions you want to import.
   Choose *Include Dependent Bill of Materials* if you want to import the material definitions with the "boms" parameter. If you do not choose this option, the material definition will not contain a reference to a BOM.
   If you choose *Include Dependent Material Classifications*, the imported material definitions will include the "classes" parameter. If you do not choose this option, the material definition will not contain a reference to classes.

   > i Note
   >
   > These options affect only the material definition and not the import of bill of material definitions or class definitions.
8. Choose *Next*.
9. Specify the folder and file name to be created to store the definitions of the imported items.

   > i Note
   >
   > Do **not** select any advanced features.
10. Choose *Finish*.

Importing Master Data to a New Project

1. In the solution modeling environment, choose *File* > *Import* > *SAP Solution Sales Configuration* > *Import Model to new Project*.
2. Choose *Next*.
3. Choose from an ECC or a CRM connection.
4. Enter the client, user, and password.
5. Choose *Next*.
6. Select the knowledge base runtime version for the product (you can sort the results by clicking any column header).

7. Select the master data definitions you want to import.
   Choose *Include Dependent Bill of Materials* if you want to import the material definitions with the “boms” parameter. If you do not choose this option, the material definition will not contain a reference to a BOM.
   If you choose *Include Dependent Material Classifications*, the imported material definitions will include the “classes” parameter. If you do not choose this option, the material definition will not contain a reference to classes.

   > i Note
   >
   > These options affect only the material definition and not the import of bill of material definitions or class definitions.

8. Choose *Next*.
9. Enter a project name.
10. Choose *Finish*.

2.2.3 Datenlader in der Lösungsmodellierungsumgebung

Verwendung

Der Datenlader ist ein Werkzeug, mit dem Sie die Konfigurationsstammdaten aus einem ECC-System (SAP ERP Central Component) oder aus einem CRM-System (Customer Relationship Management) herunterladen können. Der Datenlader wird am SAP ECC oder CRM Gateway registriert, initiiert Download-Anforderungen und verarbeitet die Rückrufe aus dem SAP-ECC- oder CRM-System. Beim Download selbst handelt es sich um einen „Push“-Mechanismus, für den zwei RFC-Verbindungen erforderlich sind:

1. eine ausgehende Verbindung vom Datenlader zum SAP-ECC- oder CRM-System, um die SAP-Gateway-Parameter zu lesen und Download-Anforderungen zu initiieren.
2. eine eingehende Verbindung vom SAP-ECC- oder CRM-System, um die „übertragenen“ Daten zu verarbeiten.

Der Datenlader wird in der Lösungsmodellierungsumgebung verwendet, um Konfigurationsstammdaten für Ihre Lösungen aus dem SAP-ECC- oder CRM-System in die Datenbank der Lösungsmodellierungsumgebung herunterzuladen. Er wird als Eclipse-Plug-In installiert. Bevor Sie Daten herunterladen können, müssen Sie eine Verbindung zum SAP-ECC- oder CRM-Quellsystem konfigurieren und dann die Verbindung den Datenlader-Konfigurationseinstellungen hinzufügen.

Weitere Informationen zur Verwendung des Datenladers in der Lösungsmodellierungsumgebung finden Sie unter **Importieren einer Wissensbasis mit dem Datenlader**.

Related Information

Importing a Model in Local Database [Seite 104]

2.3 SSC DevOps Wizard

To facilitate faster creation of automated setup for KB exports, a new SSC DevOps Project Wizard has been added in SME. You can create a new DevOps project from the file menu and select the knowledge bases and destinations for export. The wizard will generate a script for individual connections, and a master script for exporting to all the connections. The wizard uses the KB project and its dependent project information from the eclipse workspace to generate the scripts. These scripts can also be updated later to include more knowledge bases and connections.

To create a new DevOps project, perform the following steps:

1. Go to *File* > *New* > *SSC DevOps Project* and specify the *Project Name* and *Location*.
2. In the next window, select the knowledge bases to be exported, and in the subsequent window, select the connections to which the KBs need to be exported.
3. Upon finishing, one script for each individual connection, and a master script that can execute all the individual scripts will be generated.
4. This DevOps project can be executed using SSC DevOps run configurations. Results of the execution are displayed in the eclipse console.
5. To update the project, right click and select the *Update Devops Project* menu option. New SSC projects added in the workspace will be listed which can be then selected for script creation.

This DevOps project can be managed in Git and can be then used to setup headless KB export on a continuous integration server like Jenkins. For more infomartion, refer to the SSC Solution Modeling Environment Overview presentation on the http://help.sap.com. Search for *SAP Solution Sales Configuration* > *Relevant Version* > *Additional Information* > *SSC Solution Modeling Environment Overview*.

2.4 Analyzing Pricing Traces

It is now possible to turn on pricing traces while testing the configuration. The pricing context parameters can be set in preferences, and the pricing context can be set during the opening of a knowledge base in SME.

When the pricing context is enabled, pricing parameters set in preferences will be passed onto the configuration session, and pricing will be enabled to turn on the pricing traces. It needs to be ensured that the material and pricing data is already downloaded in the database using which the tests are being performed.

You need to perform the following steps to use this feature:

1. Go to *File* > *Open Knowledge Base* and select the connection.
2. On the knowledge base selection page, select the kb.
3. Click the *Pricing Context Properties* button. You will be navigated to the pricing context properties preference page. Set the pricing context properties here and click on *Apply*.

   You can also go to this page by navigating to *Window* > *Preferences* > *SAP SSC* > *Pricing Context Properties*
4. On the KB selection page, select the check box to *Set Pricing Context*. It will turn on pricing for this configuration.

5. Perform the configuration in the test perspective.
6. To view the pricing traces for the current configuration, right-click on the model tree and select *Show Pricing Traces*. This will open the pricing traces in the eclipse xml editor.
7. Similarly, to turn on pricing for Restore Configuration, we need to set the pricing context properties in the preferences, and select the *Set Pricing Context* checkbox on the restore configuration page

For more infomartion, refer to the SSC Solution Modeling Environment Overview presentation on the http://help.sap.com. Search for *SAP Solution Sales Configuration* > *Relevant Version* > *Additional Information* > *SSC Solution Modeling Environment Overview*.

2.5 Data Exchange

Use

Master data must be exchanged between the solution modeling environment and the target sales system, for example, SAP Customer Relationship Management (SAP CRM) or SAP ERP Core Component (SAP ECC). The main data exchanges are as follows:

- Material master data and compatible mode (classical IPC) knowledge bases are replicated from the SAP ECC system to the SAP CRM system.
- Modelers import data such as characteristics and classes from the SAP CRM system to the solution modeling environment.
- Modelers transfer the completed solution model (knowledge base runtime version) from the solution modeling environment to the SAP CRM system or to the SAP ECC system.

Pricing information is not exported to the solution modeling environment.

i Note

If you want to configure and export SME into a local database (for example, for use with SAP Solution Sales Configuration in Hybris), you must download and register the relevant database driver.

⚠ Caution

The Data Loader does not import product data into the solution modeling environment. Therefore, you must ensure that the products defined in the solution models also exist in the target sales system(s) where the knowledge base is deployed.

Prerequisites

You have configured the connection between the solution modeling environment and the target sales system.

More Information

For more information about importing data, see Data Loader in the Solution Modeling Environment [page 109].

2.6 Maintaining Modeling Templates

Context

You use this procedure to maintain templates for model elements. When you open an empty text file in the solution modeling environment, and then use the auto-completion feature (CTRL + SPACEBAR), the system lists all of the available templates. Templates are stored as XML files.

Procedure

1. Choose *Window* > *Preferences*

   The system displays the preferences dialog.
2. Choose *SAP Solution Sales Configuration* > *Modeling Templates*

   The system displays the *Templates* dialog.

Results

The system lists all of the existing templates. A template is defined with the following information:

- Name
- Context (for example, Bill of Material, Class, Constraint, and so on)
- Description
- Pattern

If you select a template, the content of the template is displayed in the *Preview* area.

The *Templates* dialog provides the following functions:

Function	Description
New...	You use this function to create a new template.

Function	Description
Edit...	You use this function to edit the selected template.
Remove	You use this function to remove the selected template.
Import	You use this function to import a template from an XML file.
Export	You use this function to export the selected template to an XML file.

2.7 Developing SSC User Exits

To facilitate quick development of user exits, a new wizard has been added in SME. This wizard will create a skeleton for a pFunction project in which custom classes can be added. This pFunction can be then exported from SME to the local database or backend system for testing.

While executing a configuration in SME test perspective, pFunctions would be loaded from the database along with the KB. The existing pFunction jar files can also be exported using this project by placing the jar files in the target directory of the pFunction project.

You can perform the following steps to use this wizard:

1. Go to *File* > *New* > *SSC pFunction Project*.
2. Specify the *Project Name*, *Unit Name*, and *Unit Version* and create the project.
3. A project will be created in the eclipse workspace in which java classes can be added.
4. To export the pFunction, select the pFunction project and go to *File* > *Export* > *SAP SSC* > *Export SSC pFunction*.

In case a custom unit version has been used during project creation, the same needs to be specified in the engine settings preference page before start of the configuration in the testing perspective.

For more infomartion, refer to the SSC Solution Modeling Environment Overview presentation on the http://help.sap.com. Search for *SAP Solution Sales Configuration* > *Relevant Version* > *Additional Information* > *SSC Solution Modeling Environment Overview*.

2.8 Zusammenarbeit zwischen Modelern

Verwendung

Komplexe Lösungsmodelle, in denen verschiedene Produkte kombiniert werden, werden häufig von mehreren Modelern bearbeitet. Darüber hinaus können Stammdaten in verschiedenen Produkten wiederverwendet werden. Dies bedeutet, dass unterschiedliche Modeler u.U. an denselben Modellelementen Änderungen vornehmen müssen. Mit dieser Funktion wird sichergestellt, dass lokal gepflegte Modelle zusammengeführt und synchronisiert werden können, indem ein Modell-Repository verwendet wird, das die Regeln und Workflows für die Zusammenarbeit definiert.

Wenn Änderungen an einem Lösungsmodell vorgenommen werden, wird das Modell nicht direkt im SAP-CRM- (SAP Customer Relationship Management) oder SAP-ERP-System (SAP Enterprise Resource Planning) gespeichert. Zuerst werden die Änderungen in einem Modell-Repository zusammengeführt, in dem die Regeln und Workflows für die Zusammenarbeit definiert sind. Eine konsistente Modellversion wird dann aus dem Modell-Repository an das relevante System übertragen.

> **i Note**
>
> Die Lösungsmodellierungsumgebung enthält keine eigene Repository-Technologie; stattdessen ist sie zur Verwendung mit vorhandenen, dateibasierten Quellcode-Verwaltungssystemen (SCMS) konzipiert.

Integration

Die Lösungsmodellierungsumgebung basiert auf der Eclipse-Plattform, und Connector-Plug-Ins sind für die gängigsten SCMS wie Subversion, CVS und Git frei verfügbar. Die meisten SCMS werden auch mit dem Eclipse-Connector-Plug-In ausgeliefert.

Das SCMS kann folgendermaßen mit der Lösungsmodellierungsumgebung integriert werden:

- Indem das erforderliche SCMS-Connector-Plug-In über den integrierten Update Manager in die Lösungsmodellierungsumgebung heruntergeladen wird.
- Indem die Lösungsmodellierungsumgebung als Plug-In in eine vorhandene Eclipse-integrierte Entwicklungsumgebung (IDE) mit einem vorhandenen SCMS Connector heruntergeladen wird.

2.9 Setting Context Properties

The configuration engine uses reference characteristics to obtain the context of the environment in which it is working. An example of this context could be a sales organization, a distribution channel, etc. In SAP CRM/SAP ECC/Hybris, this context is passed on to the configuration engine through RFCs and APIs.

In SME, this context can be passed using a properties file. The format of the context properties file should be:

```
ref_cstic1=value1
ref_cstic2=value2
```

You can use this procedure to set context properties while opening a runtime knowledge base version or while restoring configuration.

Setting Context Properties When Opening a KB and Creating a New Configuration

1. Choose *File* > *Open Knowledge Base*.
2. Enter the connection details.
3. Choose *Next*.
4. Select the knowledge base and profile.
5. Select *Context Property*.
6. Browse the property file.
7. Choose *Finish*.

Setting Context Properties When Restoring an Existing Configuration

1. Choose *File* > *Restore Configuration*.
2. Enter the connection details.
3. Choose *Next*.
4. Browse the configuration file.
5. Select *Context Property*.
6. Browse the property file.
7. Choose *Finish*.

2.10 Automating Modeling Lifecycle

The solution modeling environment interface allows a user to use different features of the product.

Most of the features on the SME UI require human intervention for execution, for example, exporting a model project to a target system, restoring a configuration, execution of performer on a model project, etc.

The headless automated process not only allows automation of these features, but also allows the user to run regular automated jobs to ensure stability of models. The features mentioned below are currently supported in headless automation:

- Headless Export of Knowledge Bases
- Headless Performer Execution
- Headless XML Configuration Restore
- Headless Upload of External Variant Table
- Headless Dataloader Execution

2.10.1 Headless Export of Knowledge Bases

Use

You can use this process to export solution models to a target system, for example, a standalone database, SAP ERP, or SAP CRM.

Prerequisites

- You have installed the solution modeling environment on the system from which you want to export the solution models
- You have set up an Eclipse workspace for the solution modeling environment
- You have defined the required back-end connections under *Window* > *Preferences* > *SAP Solution Sales Configuration* > *Connections*
- If solution models are to be exported to any of the supported database systems (see the Product Availability Matrix at http://support.sap.com/pam), such as Microsoft SQL Server, you have already registered the relevant database driver under *Window* > *Preferences* > *SAP Solution Sales Configuration* > *Drivers*
- You have configured all relevant settings in your solution modeling environment, such as source formatting and validation settings, under *Window* > *Preferences* > *SAP Solution Sales Configuration* > *Source Formatter / Validation*

Preparatory Steps

The knowledge base exporter requires the following information:

- The connection details of the target system
  The available connections are configured in the solution modeling environment under *Window* > *Preferences* > *SAP Solution Sales Configuration* > *Connections* and are stored under `<workspace_directory>\.metadata\.plugins\com.sap.custdev.projects.fbs.slc.conn.core\connections.xml`.
- The settings for the modeling language source formatter and model validation
  These settings are configured under *Window* > *Preferences* > *SAP Solution Sales Configuration* > *Source Formatter / Validation* and are stored under `<workspace_directory>\.metadata\.plugins\com.sap.custdev.projects.fbs.slc.sme\settings.xml`.

This information can be provided in multiple ways based on the mode of operation of the exporter:

- **Workspace mode**
  An Eclipse workspace is used to determine the settings and relevant modeling projects.
- **Directory mode**

A minimal Eclipse workspace is used, that is, it should contain only registered database drivers. The settings and modeling projects are specified by parameters that are passed to the headless exporter executable.

2.10.1.1 Ausführen des monitorlosen Exports

Um das monitorlose Exportprogramm der Wissensbasis zu starten, müssen Sie die Datei `eclipse.exe` oder die Datei `eclipsec.exe` im Eclipse-Installationsverzeichnis ausführen.

→ Recommendation

Schreiben Sie eine Batch-Datei (`.bat`) oder eine Befehlsdatei (`.cmd`), die die ausführbare Datei aufruft.

2.10.1.1.1 Eclipse-spezifische Argumente

Das Verhalten des Exports kann durch folgende Argumente gesteuert werden:

Eclipse-spezifische Argumente

Argumentname	Argumentbeschreibung
`--launcher.suppressErrors`	Meldungen und Fehlerdialogfenster unterdrücken
`-application com.sap.custdev.projects.fbs.slc.sme.export.headless.application`	Die auszuführende Anwendung Diese muss auf die monitorlose Exportanwendung zeigen.
`-noSplash`	Begrüßungsbild nicht anzeigen
`-console`	Konsolenfenster anzeigen
`-consoleLog`	Meldungen an die Konsole schreiben
`-vmargs`	Java-VM-Argumente Muss das letzte Argument sein, da alle Argumente danach als Java-VM-Argumente geparst werden. Zum Beispiel `-vmargs -Xms128M -Xmx512M`, `-XX:PermSize=128M -XX:MaxPermSize=256M` usw.

2.10.1.1.2 Export Application-Specific Argument

The behavior of the export can be controlled by the following arguments:

Export-Application-Specific Arguments

Argument Name	Example	Argument Description
`-data`	`-data "C:\path\to\workspace"`	The path to the Eclipse workspace. The workspace has to contain at least a JDBC driver registration (in directory mode). In workspace mode, the workspace also has to contain valid settings.
`-kbData`	`-kbData "C:\path\to\projects"`	(Relevant only in directory mode) The path to the directory with sub-directories containing all required `.ssc` files.
`-kbProject`	`-kbProject model.project.name`	**Workspace mode:** The name of the project that contains the KB to export. **Directory mode:** The directory names of all sub-directories in `-kbData` to be scanned. Use `-kbProject "subDir1;subDir2"`.
`-kbName`	`-kbName KB_NAME`	The name of the knowledge base to be exported
`-kbValidate`	Not applicable	If specified, the knowledge base is validated before the export. i Note Always specify this parameter to prevent inconsistent knowledge bases from being exported.
`-conFile`	`-conFile "path\to\connections.xml"`	**Workspace mode:** Optional absolute path to connections to be used instead of the connections defined in the workspace. **Directory mode:** Required path to connections to be used. It can be absolute or relative to `-kbData`. i Note You can add logsys parameter with value to the connection.xml file when LOGSYS needs to be overwritten in the headless mode.

Argument Name	Example	Argument Description
`-conName`	`-conName SYSTEM_1`	The name of the export connection to be used. If it is not specified, the export uses the default connection.
`-conPassword`	`-conPassword secret`	The password to use for the chosen connection.
`-settingsFile`	`-settingsFile "path\to\settings.xml"`	**Workspace mode:** Optional absolute path to the settings to be used instead of the settings defined in the work-space. **Directory mode:** Required path to set-tings to be used. It can be absolute or relative to `-kbData`.
`-kbConsoleLog`	Not applicable	Log messages and errors to the con-sole.
`-kbEclipseLog`	Not applicable	Log messages and errors to the Eclipse runtime logging framework.
`-kbExcludeVariantTableContents`	Not applicable	Variant tables are exported without content.
`-kbExcludeLocalizations`	Not applicable	Localized short and long texts of model elements are not exported.

2.10.1.1.3 Example: Headless Knowledge Base Export

If the headless export code is put into a command file called `deployKb.cmd`, it can be used as follows:

- **Workspace mode**
  In case of the workspace mode, a typical call to the exporter appears as follows:

```
eclipse\eclipsec -vm --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.sme.export.headless.application -noSplash
– console
-consoleLog -kbValidate -kbProject -kbName -kbConsoleLog %* -vmargs
-Xms128M -Xmx1024M - XX:PermSize=128M -XX:MaxPermSize=256M
echo EXIT CODE: %ERRORLEVEL%
```

  → Remember

  If the default eclipse workspace is not used, then `-data` argument should be used to specify the workspace path in headless export mode.

- **Directory mode**

```
eclipsec --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.sme.export.headless.application -noSplash
-console
-consoleLog -kbValidate -kbConsoleLog %* -vmargs -Xms128M -Xmx1024M -
XX:PermSize=128M -XX:MaxPermSize=256M echo EXIT CODE: %ERRORLEVEL% -data
"C:\path\to\workspace" -kbData
"C:\path\to\projects_root"
-kbProject your.project.with.kb -kbName KB_NAME -conFile
"C:\path\to\connections.xml" -settingsFile "C:\path\to\settings.xml"
```

The application returns an exit code that can be accessed through variable `ERRORLEVEL`.

For more information about this, a sample batch script is available in SAP Note 2174488 (Headless Knowledge Base Export - Sample Script).

2.10.1.1.4 Exit-Codes für Wissensbasisexport

Exit-Codes für Wissensbasisexport

Exit-Code	Beschreibung
0	Alles OK. WB wurde exportiert.
200	Ungültige oder unzureichende Argumente oder Parameter.
201	Angegebenes Projekt (`-kbProject`) nicht gefunden.
202	Angegebene WB (`-kbName`) nicht gefunden.
203	Angegebene WB ist ungültig (Validierungsprobleme).
204	Verbindung nicht gefunden oder ungültig.
205	Ausführen von Export nicht möglich (interne Probleme; siehe Protokoll).

2.10.2 Ausführung des monitorlosen Performers

Die Ausführung des monitorlosen Performers ermöglicht es den Benutzern, anhand eines automatisierten Prozesses Performer-Skripte für die exportierten Modelle im Zielsystem auszuführen.

Zum Bereitstellen von Daten für die Ausführung des monitorlosen Performers können Sie entweder das Arbeitsbereichsszenario oder das Ordnerszenario verwenden.

Arbeitsbereichsszenario

Hierbei handelt es sich um einen Eclipse-Arbeitsbereich mit einem Projekt und definierten Verbindungen sowie einem Performer-Skript-Ordner.

Beim Arbeitsbereichsszenario sind für die Ausführung des monitorlosen Performer-Skripts folgende Daten erforderlich:

- Projektarbeitsbereich mit Projekten
- Gültige definierte Verbindungen
- Ordner für Performer-Skripte (arbeitsbereichsspezifische Performer)

Der Arbeitsbereich sollte die Projekte enthalten. Zum Einrichten einer Exportverbindung wählen Sie *Fenster* › *Voreinstellungen* › *SAP-Lösungskonfiguration* › *Verbindungen*.

i Note

Wenn Sie den Arbeitsbereich schließen, werden die Verbindungen unter folgendem Pfad abgelegt:

```
<workspace_dir.metadata>\.plugins\com.sap.custdev.projects.fbs.slc.conn.core\conn
ections.xml
```

⚠ Caution

Fügen Sie der XML-Datei auf eigenes Risiko ein Verbindungskennwort hinzu. Das Kennwort wird im unsicheren Nur-Text-Format gespeichert. Fügen Sie hierzu dem Element `clientSettings` das Attribut `password` hinzu.

Ordnerszenario

Hierbei handelt es sich um ein einfaches Verzeichnis, das Unterverzeichnisse mit den `.performer`-Skriptdateien und den erforderlichen PFunctions (sofern vorhanden) in den entsprechenden, durch Parameter angegebenen Verzeichnissen und Pfaden zu einer gültigen `connection.xml`-Datei enthält.

Beim Ordnerszenario sind für die Ausführung des monitorlosen Performer-Skripts folgende Daten erforderlich:

- Ein oder mehrere Ordner als Container für die `.performer`-Skriptdateien
- Eine XML-Datei für die zu verwendende Performer-Verbindung (Datenbank) (`connections.xml`)
- Pfad zur `.jar`-Datei des Verbindungstreibers (z.B. `D:\jars\sqljdbc4.jar`).

Die Anwendung benötigt einen Pfad zu einem (übergeordneten) Ordner (`-psData “dir/parent”`), und sie durchsucht sämtliche Unterordner nach den verfügbaren Performer-Skripten. Dieses Szenario erfordert außerdem einen Pfad zu den Verbindungen über eine XML-Datei (`-conFile “path/connections.xml”`).

Sowohl im Arbeitsbereichs- als auch im Ordnerszenario muss für den Performer-Lauf die Datei `pFunction.jar` im selben Ordner wie das entsprechende Skript enthalten sein.

2.10.2.1 Ausführen des monitorlosen Performers

Zum Ausführen des „monitorlosen Performer-Skripts“ rufen Sie die Datei **`eclipse.exe`** oder die Datei **`eclipsec.exe`** (ohne zusätzliche Konsole) im Eclipse-Installationsordner mit den erforderlichen Argumenten auf.

→ Recommendation

Wir empfehlen Ihnen, eine Batch-Datei (`.bat`) oder eine Befehlsdatei (`.cmd`) zu verwenden.

2.10.2.1.1 Eclipse-spezifische Argumente

Argument	Beschreibung
--launcher.suppressErrors	Das Argument zum Unterdrücken von Fehlern muss vor `vmargs` gesetzt werden; es handelt sich in der Regel um eines der ersten Argumente.
-application com.sap.custdev.projects.fbs.slc.config.performer.headless.performerapplication	Die zu startende Anwendung. Der Wert (Anwendungs-ID) wird durch ein Leerzeichen vom Argument getrennt.
-noSplash	Begrüßungsbild nicht anzeigen
-console	Konsole anzeigen
-consoleLog	Protokoll für die Konsole schreiben
-vmargs	Java-VM-Argumente. Dies muss das letzte Eclipse-Argument sein, da alle folgenden Argumente als Java-VM-Argumente geparst werden, z.B.: `-vmargs -Xms128M -Xmx512M` `-XX:PermSize=128M -XX:MaxPermSize=256M`

i Posting Instructions

Eine vollständige Liste der unterstützten Argumente finden Sie in der Eclipse-Dokumentation .

2.10.2.1.2 Performer-Application-Specific Arguments

i Note

Use quotation marks ("") for values that contain spaces such as `-psdata “D: \Performer Data”`.

Parameter	Workspace Scenario	Directory Scenario
-projectWorkspace	Workspace project name. -projectWorkspace ssc_office_example	Not applicable
-psData	The path to the directory with the subdirectories containing all required `.performer` files. -psData “C:\parentDirectory”	The path to the directory with the subdirectories containing all required `.performer` files. -psData “C:\parentDirectory”
-conFile	Not applicable	Required path to connections to use. -conFile “C:\path\connections.xml”
-conDriver	Not applicable	The path to the connection driver jar file. -conDriver “D:\jars\sqljdbc4.jar”
-psConsoleLog	Print logs on console. -consoleLog	Print logs on console. -consoleLog
-conName	Not applicable	Connection name. -conName “localCon”
-conPassword	Connection password (optional if specified in `connections.xml` file). -conPassword “mypassword”	Connection password (optional here if specified in `connections.xml` file). -conPassword “mypassword”
-pContextPropertyFile i Note Since 3.0 Patch 6, the context properties are saved in a performer file which can also be used in the headless execution mode.	Path for the context properties to be set. This is an optional field and can be used as required by user. For example, C:\Users\Admin\ContextProperty\sme_office_prop.properties.	Path for the context properties to be set. This is an optional field and can be used as required by user. For example, C:\Users\Admin\ContextProperty\sme_office_prop.properties.
-pFunction	Specify the required pfunctions(if any). For example, -pFunction "C:/users/dir1/pFunction1.jar;C:/users/dir2/pFunction2.jar".	Specify the required pfunctions(if any). For example, -pFunction "C:/users/dir1/pFunction1.jar;C:/users/dir2/pFunction2.jar".

2.10.2.1.3 Beispiel: Ausführung des monitorlosen Performers

→ Recommendation

Wir empfehlen Ihnen, für diesen Prozess eine Batch-Datei (`performer.bat`) zu verwenden.

Die Anwendung gibt einen Exit-Code zurück (siehe Echo `%ERRORLEVEL%` im Beispielbefehl).

Arbeitsbereichsmodus

Ein Batch- oder ein Befehlsskript kann folgendermaßen aussehen:

Sample Code

```
@echo off
eclipse\eclipsec -vm “%JRE_HOME%\bin” --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.config.performer.headless.performerapplicatio
n -noSplash -console -psConsoleLog -projectWorkspace “ssc_office_example”
-psData “C:\Users\Public\PROJECTWORK\Enhancements\TestRunner in Headless
Mode\PerformerData” -conName “localCon” -conPassword “mypassword”
```

Ordnermodus

Ein Batch- oder ein Befehlsskript kann folgendermaßen aussehen:

Sample Code

```
@echo off
eclipse\eclipsec -vm "%JRE_HOME%\bin" --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.config.performer.headless.performerapplicatio
n -noSplash -console -psConsoleLog -conDriver
"C:\Users\Public\SOFTWARES\sqljdbc4.jar" –conFile
"C:\Users\Public\PROJECTWORK\Enhancements\TestRunner in Headless
Mode\PerformerData\config\connections.xml" –psData
"C:\Users\Public\PROJECTWORK\Enhancements\TestRunner in Headless
Mode\PerformerData"REM -Xdebug -Xnoagent -Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005@echo EXIT CODE:
%ERRORLEVEL%
```

2.10.2.1.4 Exit-Codes für Performer-Ausführung

Exit-Codes für Performer-Ausführung

Exit-Code	Beschreibung
0	OK. Performer-Skripte wurden ausgeführt.
300	Ungültige oder unzureichende Argumente oder Parameter.
301	Angegebene Performer-Skripte (`-psData`) nicht gefunden.
302	Angegebenes Arbeitsbereichsprojekt nicht gefunden.
303	Wissensbasis nicht gefunden.
304	Verbindung nicht gefunden oder ungültig.
305	Ausführen von Performer-Skripten nicht möglich (interne Probleme; siehe Protokoll).

2.10.3 Monitorlose XML-Konfigurationswiederherstellung

Die Funktion zur monitorlosen Konfigurationswiederherstellung ist der Bestandteil der Lösungsmodellierungsumgebung, der die Konfigurationswiederherstellung mit den .cfg-/.xml-Dateien über den monitorlosen automatisierten Prozess für die exportierten Modelle im Zielsystem ermöglicht.

Zur Bereitstellung der Daten für die monitorlose Konfigurationswiederherstellung stehen zwei Szenarien zur Verfügung:

- **Arbeitsbereichsszenario**
  Ein Eclipse-Arbeitsbereich mit Projekt und dessen definierten Verbindungen. Die monitorlose Konfigurationswiederherstellung erfordert folgende Daten für das Arbeitsbereichsszenario:
  - Projektarbeitsbereich mit Projekten
  - Gültige definierte Verbindungen
  - Konfigurationsdateiverzeichnis (enthält .cfg-/.xml-Dateien)

  Der Arbeitsbereich sollte alle relevanten Projekte umfassen. Zum Einrichten einer Exportverbindung wählen Sie *Fenster* > *Voreinstellungen* > *SAP-Lösungskonfiguration* > *Verbindungen*.

  > i Note
  >
  > Wenn Sie den Arbeitsbereich schließen, werden die Verbindungen unter folgendem Pfad abgelegt:
  >
  > `<workspace_dir.metadata>\.plugins\com.sap.custdev.projects.fbs.slc.conn.core\connections.xml`

  > ⚠ Caution
  >
  > Fügen Sie der XML-Datei auf eigenes Risiko ein Verbindungskennwort hinzu. Das Kennwort wird im unsicheren Nur-Text-Format gespeichert. Fügen Sie hierzu dem Element `clientSettings` das Attribut `password` hinzu.

- **Verzeichnisszenario**
  Ein einfaches Verzeichnis mit Unterverzeichnissen mit den .cfg-/.xml-Dateien und Pfaden, wie durch Parameter für eine gültige `connection.xml` angegeben. Zur Ausführung des Skripts für die monitorlose Konfigurationswiederherstellung sind folgende Daten für das Verzeichnisszenario erforderlich:
  - Konfigurationsdateiverzeichnis (enthält .cfg-/.xml-Dateien)
  - XML-Datei (`connections.xml`) für die Verbindung (DB)
  - Pfad zur jar-Datei des Verbindungstreibers, z.B. `D:\jars\sqljdbs4.jar`

  Die Anwendung benötigt einen Pfad zu einem übergeordneten Verzeichnis (`-rsData "dir/parent"`), und sie durchsucht sämtliche Unterverzeichnisse nach den verfügbaren Konfigurationsdateien (.cfg/.xml). Dieses Szenario erfordert außerdem einen Pfad zu den Verbindungen über eine XML-Datei (`-conFile "path/connections.xml"`).

2.10.3.1 Ausführen der monitorlosen XML-Konfigurationswiederherstellung

Um das Skript zur monitorlosen Konfigurationswiederherstellung auszuführen, rufen Sie die Datei `eclipse.exe` oder `eclipsec.exe` (ohne zusätzliche Konsole) im Eclipse-Installationsverzeichnis mit den erforderlichen Argumenten auf.

> → Recommendation
>
> Schreiben Sie eine Batch-Datei (`.bat`) oder eine Befehlsdatei (`.cmd`), die die ausführbare Datei aufruft.

2.10.3.1.1 Eclipse-spezifische Argumente

Das Verhalten des Exports kann durch folgende Argumente gesteuert werden:

Eclipse-spezifische Argumente

Argumentname	Argumentbeschreibung
`--launcher.suppressErrors`	Das Unterdrücken von Fehlern und Argumenten muss vor `vmargs` festgelegt werden, da es eines der ersten Argumente ist.
`-application com.sap.custdev.projects.fbs.slc.config.restore.headless.restoreapplication`	Die zu startende Anwendung. Der Wert (Anwendungs-ID) wird durch ein einzelnes Leerzeichen von dem Argument getrennt.
`-noSplash`	Begrüßungsbild nicht anzeigen
`-console`	Konsole anzeigen

Argumentname	Argumentbeschreibung
`-consoleLog`	An der Konsole anmelden.
`-vmargs`	Java-VM-Argumente. Sollte das letzte Eclipse-Argument sein, da alle Argumente danach als Java-VM-Argumente geparst werden. Zum Beispiel `-vmargs -Xms128M -Xmx512M`, `-XX:PermSize=128M -XX:MaxPermSize=256M` usw.

i Posting Instructions

Eine vollständige Liste der unterstützten Argumente finden Sie in der Eclipse-Dokumentation .

2.10.3.1.2 Anwendungsspezifische Argumente für die Konfigurationswiederherstellung

Das Verhalten des Exports kann durch folgende Argumente gesteuert werden:

Anwendungsspezifische Argumente für die Konfigurationswiederherstellung

Parameter	Arbeitsbereichsszenario	Verzeichnisszenario
`-projectWorkspace`	Name des Arbeitsbereichsprojekts `-projectWorkspace ssc_office_example`	Nicht anwendbar
`-rsData`	Der Pfad zu dem Verzeichnis mit den Unterverzeichnissen, die alle erforderlichen Konfigurationsdateien enthalten `-rsData “C:\parentDirectory”`	Der Pfad zu dem Verzeichnis mit den Unterverzeichnissen, die alle erforderlichen Konfigurationsdateien enthalten `-rsData “C:\parentDirectory”`
`-conFile`	Nicht anwendbar	Erforderlicher Pfad zu den relevanten Verbindungen. `-conFile “C:\path\connections.xml”`
`-conDriver`	Nicht anwendbar	Pfad zur .jar-Datei des Verbindungstreibers. `-conDriver “D:\jars\sqljdbc4.jar”`

Parameter	Arbeitsbereichsszenario	Verzeichnisszenario
`-rsConsoleLog`	Protokolle für Konsole drucken. `- rsConsoleLog`	Protokolle für Konsole drucken. `- rsConsoleLog`
`-conName`	Nicht anwendbar	Verbindungsname. `-conName “localCon”`
`-conPassword`	Verbindungskennwort (optional, wenn nicht in der connections.xml-Datei angegeben). `-conPassword “mypassword”`	Verbindungskennwort (optional, wenn nicht in der connections.xml-Datei angegeben). `-conPassword “mypassword”`

i Note

“” werden für Werte mit Leerzeichen verwendet, zum Beispiel `-rsdata “D: \Configuration Data”`.

2.10.3.1.3 Beispiel: Monitorlose XML-Konfigurationswiederherstellung

Es wird empfohlen, die Batch-Datei `restoreConfig.bat` für den Wiederherstellungsprozess zu verwenden; die Anwendung gibt einen Exit-Code zurück, nachdem die Wiederherstellung abgeschlossen ist.

Weitere Informationen hierzu finden Sie unter `echo %ERRORLEVEL%` im Beispielbefehl.

Arbeitsbereichsmodus

```
eclipse\eclipsec -vm "%JRE_HOME%\bin" --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.config.restore.headless.restoreapplication
-noSplash -console -rsConsoleLog  -projectWorkspace "ssc_office_example" -rsData
"C:\Users\Demo\PROJECTWORK\Enhancements\RestoreConfigData" -conName "localCon"
-conPassword "mypassword"
```

Verzeichnismodus

```
@echo off
eclipse\eclipsec -vm "%JRE_HOME%\bin" --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.config.restore.headless.restoreapplication
-noSplash -console -rsConsoleLog
-conDriver "C:\Users\Demo\SOFTWARES\sqljdbc4.jar" –conFile
"C:\Users\Demo\PROJECTWORK\Enhancements\RestoreConfigData\config\connection.xml"
–rsData "C:\Users\Demo\PROJECTWORK\Enhancements\ RestoreConfigData"
REM -Xdebug -Xnoagent -Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005 @echo EXIT CODE:
%ERRORLEVEL%
```

2.10.3.1.4 Exit-Codes für Konfigurationswiederherstellung

Exit-Code	Beschreibung
0	Alles OK. Konfigurationsdateien wiederhergestellt.
300	Ungültige oder unzureichende Argumente oder Parameter.
301	Angegebene Konfigurationsdateien (`-rsData`) nicht gefunden.
302	Angegebenes Arbeitsbereichsprojekt nicht gefunden.
303	Wissensbasis nicht gefunden.
304	Verbindung nicht gefunden oder ungültig.
305	Konfiguration kann nicht wiederhergestellt werden (interne Probleme; siehe Protokoll).
306	Konfiguration kann nicht wiederhergestellt werden (interne Probleme; siehe Protokoll).
307	Datenbanktreiber nicht gefunden (interne Probleme; siehe Protokoll).

2.10.4 Monitorloses Hochladen externer Variantentabellen

Die Funktion für monitorloses Hochladen externer Variantentabellen ist der Bestandteil der Lösungsmodellierungsumgebung, der das Hochladen externer Variantentabellen in ein Zielsystem über einen monitorlosen automatisierten Prozess ermöglicht.

Zur Bereitstellung der Daten für das monitorlose Hochladen externer Variantentabellen stehen zwei Szenarien zur Verfügung:

- **Arbeitsbereichsszenario**
  Ein Eclipse-Arbeitsbereich mit einem Projekt und dessen definierten Verbindungen. Für das monitorlose Hochladen externer Variantentabellen sind folgende Daten für das Arbeitsbereichsszenario erforderlich:
  - Projektarbeitsbereich mit Projekten
  - Gültige definierte Verbindungen
  - Datenverzeichnis für externe Variantentabelle, das aus dem System `ECC/CRM` heruntergeladen wurde.

  Der Arbeitsbereich sollte alle relevanten Projekte umfassen. Zum Einrichten einer Exportverbindung wählen Sie *Fenster* > *Voreinstellungen* > *SAP-Lösungskonfiguration* > *Verbindungen*.

i Note

Wenn Sie den Arbeitsbereich schließen, werden die Verbindungen unter folgendem Pfad abgelegt:

`<workspace_dir.metadata>\.plugins\com.sap.custdev.projects.fbs.slc.conn.core\connections.xml`

⚠ Caution

Fügen Sie der XML-Datei auf eigenes Risiko ein Verbindungskennwort hinzu. Das Kennwort wird im unsicheren Nur-Text-Format gespeichert. Fügen Sie hierzu dem Element `clientSettings` das Attribut `password` hinzu.

- **Verzeichnisszenario**
  Ein einfaches Verzeichnis, das die Datendateien der externen Variantentabellen und deren Pfade enthält, wie durch Parameter für eine gültige `connection.xml` angegeben. Zur Ausführung des Skripts für die monitorlose externe Variantentabelle sind folgende Daten für das Verzeichnisszenario erforderlich:
  - Datenverzeichnis für externe Variantentabelle, das aus dem ECC-/CRM-System heruntergeladen wurde
  - XML-Datei (`connections.xml`) für die Verbindung (DB)
  - Pfad zur jar-Datei des Verbindungstreibers, z.B. `D:\jars\sqljdbc4.jar`

Die Anwendung benötigt einen Pfad zu einem übergeordneten Verzeichnis (`-vtData “dir/parent”`), die aus dem ECC-/CRM-Download-Hilfsprogramm für externe Variantentabellen heruntergeladenen Daten. Dieses Szenario erfordert außerdem einen Pfad zu den Verbindungen über eine XML-Datei (`-conFile “path/connections.xml”`).

2.10.4.1 Ausführen des monitorlosen Hochladens externer Variantentabellen

Um das Skript zum monitorlosen Hochladen externer Variantentabellen auszuführen, rufen Sie die Datei `eclipse.exe` oder `eclipsec.exe` (ohne zusätzliche Konsole) im Eclipse-Installationsverzeichnis mit den erforderlichen Argumenten auf.

→ Recommendation

Schreiben Sie eine Batch-Datei (`.bat`) oder eine Befehlsdatei (`.cmd`), die die ausführbare Datei aufruft.

2.10.4.1.1 Eclipse-spezifische Argumente

Das Verhalten des Exports kann durch folgende Argumente gesteuert werden:

Eclipse-spezifische Argumente

Argumentname	Argumentbeschreibung
`-launcher.suppressErrors`	Das Argument zum Unterdrücken von Fehlern muss vor vmargs gesetzt werden; es handelt sich in der Regel um eines der ersten Argumente.
`-application com.sap.custdev.projects.fbs.slc.sme.extvartable.upload.headless.externalvarianttableapplication`	Die zu startende Anwendung. Der Wert (Anwendungs-ID) wird durch ein Leerzeichen vom Argument getrennt.
`-noSplash`	Begrüßungsbild nicht anzeigen
`-console`	Konsole anzeigen
`consoleLog`	An der Konsole anmelden.
`-vmargs`	Java-VM-Argumente Muss das letzte Eclipse-Argument sein, da alle Argumente danach als Java-VM-Argumente geparst werden. Zum Beispiel `-vmargs-Xms128M-Xmx512M`, `-XX:PermSize=128` usw.

i Posting Instructions

Eine vollständige Liste der unterstützten Argumente finden Sie in der Eclipse-Dokumentation .

2.10.4.1.2 Anwendungsspezifische Argumente für das Hochladen externer Variantentabellen

Das Verhalten des Exports kann durch folgende Argumente gesteuert werden:

Anwendungsspezifische Argumente für das Hochladen externer Variantentabellen

Parameter	Arbeitsbereichsszenario	Verzeichnisszenario
-projectWorkspace	Name des Arbeitsbereichsprojekts. -projectWorkspace ssc_office_example	Nicht anwendbar
-vtData	Der Pfad zu dem Verzeichnis mit den Unterverzeichnissen, die alle erforderlichen Konfigurationsdateien enthalten. -vtData “C:\extVarDirectory”	Der Pfad zu dem Verzeichnis mit den Unterverzeichnissen, die alle erforderlichen Dateien der externen Variantentabellen enthalten. -vtData “C:\extVarDirectory”

Parameter	Arbeitsbereichsszenario	Verzeichnisszenario
-conFile	Nicht anwendbar	Erforderlicher Pfad zu den relevanten Verbindungen. -conFile “C:\path\connections.xml”
-conDriver	Nicht anwendbar	Pfad zur .jar-Datei des Verbindungstreibers. -conDriver “D:\jars\sqljdbc4.jar”
-vtConsoleLog	Protokolle für Konsole drucken. -vtConsoleLog	Protokolle für Konsole drucken. -vtConsoleLog
-conName	Nicht anwendbar	Verbindungsname. -conName “localCon”
-conPassword	Verbindungskennwort (optional, wenn in Datei `connections.xml` angegeben). -conPassword “mypassword”	Verbindungskennwort (hier optional, wenn in Datei `connections.xml` angegeben). -conPassword “mypassword”

i Note

“” werden für Werte mit Leerzeichen verwendet, zum Beispiel `-vtdata “D: \External Variant Data”`.

2.10.4.1.3 Beispiel: Monitorloses Hochladen externer Variantentabellen

Es wird empfohlen, die Batch-Datei `restore Config.bat` für den Wiederherstellungsprozess zu verwenden; die Anwendung gibt einen Exit-Code zurück, nachdem die Wiederherstellung abgeschlossen ist.

Weitere Informationen hierzu finden Sie unter `echo %ERRORLEVEL%` im Beispielbefehl.

Arbeitsbereichsmodus

```
eclipse\eclipsec -vm "%JRE_HOME%\bin" --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.sme.extvartable.upload.headless.externalvariantt
ableapplication -noSplash -console -vtConsoleLog  -projectWorkspace
"ssc_office_example" -vtData
"C:\Users\Demo\PROJECTWORK\Enhancements\ExternalVariantTableData" -conName
"localCon" -conPassword "mypassword"
```

Verzeichnismodus

```
@echo off
eclipse\eclipsec -vm "%JRE_HOME%\bin" --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.sme.extvartable.upload.headless.externalvariantt
```

```
ableapplication -noSplash -console -vtConsoleLog -conDriver
"C:\Users\Demo\SOFTWARES\sqljdbc4.jar" –conFile
"C:\Users\Demo\PROJECTWORK\Enhancements\ExtVarTabData\config\connection.xml" –
vtData "C:\Users\Demo\PROJECTWORK\Enhancements\ ExtVarTabData" REM -Xdebug
-Xnoagent -Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005 @echo EXIT CODE:
%ERRORLEVEL%
```

2.10.4.1.4 Exit-Codes für externe Variantentabellen

Exit-Codes für externe Variantentabellen

Exit-Code	Beschreibung
0	Alles OK. Externe Variantentabellendateien wiederhergestellt.
300	Ungültige oder unzureichende Argumente oder Parameter.
301	Angegebene Datendateien (`-vtData`) nicht gefunden.
302	Angegebenes Arbeitsbereichsprojekt nicht gefunden.
303	Wissensbasis nicht gefunden.
304	Verbindung nicht gefunden oder ungültig.
305	Ausführen von Skript für externe Tabelle nicht möglich (interne Probleme; siehe Protokoll).
306	Externe Variantentabellen mit Fehlern hochgeladen.
307	Datenbanktreiber nicht gefunden (interne Probleme; siehe Protokoll)

2.10.5 Headless Upload of Knowledge Bases

The process of headless upload of knowledge bases allows upload of `.xml` files pertaining to a certain knowledge base into a target system through headless automated process.

There are two possible scenarios to provide the data for headless upload of knowledge bases:

- **Workspace scenario**
  An Eclipse workspace with a project and its defined connections. The headless upload external variant table needs the following data for workspace scenario:

- Project workspace with projects
- Valid defined connections
- `.xml` files data directory for external variant table downloaded from `ECC/CRM` system.

The workspace should contain all the relevant projects. You can setup an export connection from *Window* › *Preferences* › *SAP Solution Configuration* › *Connections*.

> i Note
>
> When you close the workspace, the connections are stored under:
>
> `<workspace_dir.metadata>\.plugins\com.sap.custdev.projects.fbs.slc.conn.core\connections.xml`

> ⚠ Caution
>
> Add a connection password to the XML at your own risk. The password is stored as unsecure plain text. To do so, add the `password` attribute to the `clientSettings` element.

- **Directory scenario**
  A simple directory containing `.xml` files and their paths, given by parameters to a valid `connection.xml`. The headless upload of knowledge base script execution needs the following data for directory scenario:
  - `.xml` files directory downloaded from ECC/CRM system
  - An XML (`connections.xml`) file for the connection (DB)
  - A path to the connection driver jar file, for example, `D:\jars\sqljdbc4.jar`

The application needs a path to a parent directory (`-kbData` absolute path to the `.xml` files), the data downloaded from ECC/CRM by download utility. The scenario also requires a path to the connections via XML file (`-conFile "path/connections.xml"`).

2.10.5.1 Ausführen des monitorlosen Uploads

Um das Skript zum monitorlosen Hochladen von Wissensbasen auszuführen, rufen Sie die Datei `eclipse.exe` oder `eclipsec.exe` (ohne zusätzliche Konsole) in Ihrem Eclipse-Installationsverzeichnis mit den erforderlichen Argumenten auf.

> → Recommendation
>
> Schreiben Sie eine Batch-Datei (`.bat`) oder eine Befehlsdatei (`.cmd`), die die ausführbare Datei aufruft.

2.10.5.1.1 Eclipse-Specific Arguments

The behavior of the export can be controlled by the following arguments:

Eclipse-Specific Arguments

Argument Name	Argument Description
`--launcher.suppressErrors`	Suppress errors Arguments must be set before `vmargs` (normally one of the first arguments)
`-application com.sap.custdev.projects.fbs.slc.sme.extvartable.upload.headless.externalvarianttableapplication`	The application to run Value (application ID) is separated from the argument by a single space.
`-noSplash`	Do not show the splash screen
`-console`	Show a console window
`-consoleLog`	Log messages to the console
`-vmargs`	Java VM arguments. This must be the last Eclipse argument because all arguments after this one will be parsed as Java VM arguments. For example, `-vmargs -Xms128M -Xmx512M`, `-XX:PermSize=128M -XX:MaxPermSize=256M`, etc.

i Posting Instructions

For a full list of the supported arguments, see the Eclipse documentation .

2.10.5.1.2 Headless Upload-Specific Arguments

i Note

Use quotation marks ("") for values that contain spaces such as `-kbData “D: \KnowledgeBaseData”`.

Parameter	Workspace Scenario	Directory Scenario
-projectWorkspace	Workspace project name. -projectWorkspace ssc_office_example	Not applicable
-conFile	Not applicable	Path for connections to be used -conFile "C:\path\connections.xml"
-conDriver	Not applicable	Path to the connection driver jar file -conDriver "D:\jars\sqljdbc4.jar"

Parameter	Workspace Scenario	Directory Scenario
-uploadConsoleLog	Print logs on console -uploadConsoleLog	Print logs on console.
-conName	Not applicable	Connection name. -conName “localCon”
-conPassword	Connection password (optional if specified in `connections.xml` file). -conPassword “mypassword”	Connection password (optional here if specified in `connections.xml` file). -conPassword “mypassword”
-kbData	Path to the directory containing `.xml`files	The argument mentions the path to the directory containing `.xml` files of the knowledge base

2.10.5.1.3 Beispiel: Monitorloser Wissensbasis-Upload

→ Recommendation

Wir empfehlen Ihnen, für diesen Prozess eine Batch-Datei (`restoreConfig.bat`) zu verwenden.

Die Anwendung gibt einen Exit-Code zurück (siehe Echo `%ERRORLEVEL%` im Beispielbefehl).

Arbeitsbereichsmodus

Ein Batch- oder ein Befehlsskript kann folgendermaßen aussehen:

Sample Code

```
eclipse\eclipsec -vm "%JRE_HOME%\bin" --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.sme.uploadKb.headless.uploadapplication
-noSplash– console -consoleLog -
uploadConsoleLog  -projectWorkspace "ssc_office_example" -kbData
"C:\Users\Administrator\xmlModels\TestModel " -conName "localCon"
-conPassword "mypassword"

echo

EXIT CODE: %ERRORLEVEL%
```

Ordnermodus

Sample Code

```
@echo off

eclipse\eclipsec -vm "%JRE_HOME%\bin" --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.sme.uploadKb.headless.uploadapplication
-noSplash -console -
uploadConsoleLog -conDriver "C:\Users\Demo\SOFTWARES\sqljdbc4.jar" –conFile

"C:\Users\Demo\PROJECTWORK\Enhancements\configconnection.xml" –kbData
"C:\Users\Administrator\xmlModels\TestModel "

REM -Xdebug -Xnoagent -Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005

@echo EXIT CODE: %ERRORLEVEL%
```

2.10.5.1.4 Exit-Codes

Exit-Codes zum Hochladen von Wissensbasen

Exit-Code	Beschreibung
0	OK. WB wurde exportiert.
301	Ungültige oder unzureichende Argumente/Parameter
302	Es wurde kein Projekt gefunden.
303	Verbindung nicht gefunden oder ungültig.
304	Upload-Fehler
305	Der Datenbanktreiber wurde nicht gefunden.

2.10.6 Headless Dataloader Execution

Use

Some scenarios required a data download from SAP backend system into the local DB on frequent basis and thus manual trigger of data loader is sub optimal. In such situations, data download can be automated by running SME data loader in the headless mode.

Installation

The Headless Dataloader is part of SME and will be installed with SME dataloader feature installation. Hence there is no additional step required.

Setup of Headless Dataloader Execution

Before setup of the SME in headless mode, the dataloader in SME must be setup and initial download triggered from the UI which can help verify if dataloader is working as desired. Files generated as part of UI setup can then be used to trigger dataloader in headless mode.

Setup of headless mode requires that the absolute path for the settings file is known and provided in the scripts. The following are paths for the files:

- *Datalaoder.xml*. This is present in D:\DBdrivers\dataloader.xml directory. This file is generated in your eclipse workspace when you add dataloader connection settings in eclipse preferences. The same file can be used.
- The jdbc driver jar file for the destination database, for example, *D:\Database Drivers\mssql\sqljdbc4-4.0.jar*

Executing the Headless Eclipse Application

To execute SME in the headless mode, *eclipse.exe* or *eclipsec.exe* (without additional console) must be invoked in the Eclipse installation directory with the required arguments. This command can be written in a batch script *(.bat/.cmd)*.

Eclipse Specific Arguments

-launcher.suppressErrors	Suppress errors, arguments must set before vmargs, normally one of the first arguments
-application com.sap.custdev.projects.fbs.slc.dataloader.headless.dataloaderapplication	The application to start. Value (application ID) is separated by a space from the argument.

-noSplash	Does not show the splash screen
-console	Displays a console window
-consoleLog	Log messages to the console
-vmargs	Java VM arguments. Must be the last Eclipse argument because all following arguments will be parsed as Java VM arguments, eg: *-vmargs -Xms256m -Xmx1024m* *XX:PermSize=128M -XX:MaxPermSize=256M*

A full list of supported arguments can be found here:http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fmisc%2Fruntime-options.html

Application Specific Arguments

i Note

use *“”* for values containing spaces like *conDriver D:\Database Drivers\mssql\sqljdbc4-4.0.jar*.

Parameters	**Sample Value**
-conFile	Dataloader settings file to use. For example conFile *D:\DBdrivers\dataloader.xml* You can find dataloader settings xml file created from SME inside eclipse workspace for example, *<Workspace>\.metadata\.plugins\com.sap.custdev.projects.fbs.slc.dataloader.core*
-conDriver	JDBC driver jar file path. For e.g. conDriver *D:\Database Drivers\mssql\sqljdbc4-4.0.jar*
-psConsoleLog	Prints logs on console. *psConsoleLog*
-conName	Display name of the dataloader connection setting to be used. You must specify the following attribute from dataloader connection file. *displayName="con"* conName *con*

Parameters	Sample Value
-conPassword	Password for the source backend system is specified in the dataloader connection setting. conPassword *mypassword*
-backendPassword	Password for the user specified for source backend system in the dataloader connection settings. backendPassword *backendpassword*
-createTables	Specify this argument if create table needs to be run while running dataloader. There is no value required for this argument. To run the full initial download, it is recommended to specify this argument. createTables
-initialDownload	Dataloader runs in initial and delta download mode. In SME only initial mode is supported. Hence *initialDownload* must be specified. No value is required for this argument.
-eccCfgExtractorFlag	This flag enables download from *Comm Tables* and must be specified with true argument. -eccCfgExtractorFlag *TRUE*

Example

The usage of a batch file (dataloader.bat) file is recommended.

The application will return an exit code *(see echo %ERRORLEVEL% in example cmd)*.

A batch/command script could look like:

```
@echo off

eclipse\eclipsec -vm "%JRE_HOME%\bin" --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.dataloader.headless.dataloaderapplication -noSplash -console
-psConsoleLog -conDriver "D:\DBdrivers\Database_Drivers\mssql\sqljdbc4-4.0.jar" -conName "con"
-conPassword "password" -backendPassword "bkpassword" -conFile "D:\DBdrivers\dataloader.xml"
-createTables -initialDownload

REM -Xdebug -Xnoagent -Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005

@echo EXIT CODE: %ERRORLEVEL%
```

Exit Codes

0	All ok, Dataloader executed successfully
303	Dataloader settings file not found. Please Specify conFile
304	Issues while loading dataloader settings
307	Database driver not found
308	BackendPassword not specified
309	ConPassword not specified

2.10.6.1 Executing Headless Dataloader

To execute the script for headless dataloader, you must call the `eclipse.exe` or `eclipsec.exe` (without additional console) in your eclipse installation directory, with the required arguments.

→ Recommendation

Write a batch (`.bat`) or command (`.cmd`) file that calls the executable.

2.10.6.1.1 Eclipse-Specific Arguments

Argument Name	Argument Description
`--launcher.suppressErrors`	Suppress errors Arguments must be set before `vmargs` (normally one of the first arguments)
`-application com.sap.custdev.projects.fbs.slc.sme.extvartable.upload.headless.externalvarianttableapplication`	The application to run Value (application ID) is separated from the argument by a single space.
`-noSplash`	Do not show the splash screen
`-console`	Show a console window
`-consoleLog`	Log messages to the console

Argument Name	Argument Description
`-vmargs`	Java VM arguments. This must be the last Eclipse argument because all arguments after this one will be parsed as Java VM arguments. For example, `-vmargs -Xms128M -Xmx512M`, `-XX:PermSize=128M -XX:MaxPermSize=256M`, etc.

> **i Posting Instructions**
>
> For a full list of the supported arguments, see the Eclipse documentation .

2.10.6.1.2 Dataloader Application-Specific Arguments

> **i Note**
>
> Use quotation marks ("") for values that contain spaces such as `-conDriver "D:\Database Drivers\mssql\sqljdbc4-4.0.jar`.

Argument Name	Example	Argument Description
`-conFile`	`-conFile "D:\DBdrivers\dataloader.xml"`	The required path to the dataloader settings file to use.
`-conDriver`	`-conDriver "D:\DBdrivers\Database_Drivers \mssql\sqljdbc4-4.0.jar"`	The path to the connection driver `jar` file.
`-psConsoleLog`	`-psConsoleLog`	Print logs on console.
`-conName`	`-conName "con"`	Display name of the dataloader connection setting to be used.
`-conPassword`	`-conPassword "mypassword"`	Connection password for the destination database to be used for data download.
`-backendPassword`	`-backendPassword "backendpassword"`	Password for the user specified for source backend system in the dataloader connection settings.
`-createTables`	`-createTables`	Specify this argument if create table needs to be run while running the dataloader. There is no value required for this argument.

Argument Name	Example	Argument Description
`-initialDownload`	`-initialDownload`	Specify this argument if initial download needs to be run while running the dataloader. No value required for this argument.
`-eccCfgExtractorFlag`	`-eccCfgExtractorFlag “TRUE”`	This argument is used to specify the value for ECC cfg extractor flag from SME. (com.sap.sxe.dataloader.GET_KB_FROM_CFGEXTRACTOR)

2.10.6.1.3 Example: Headless Dataloader Execution

→ Recommendation

We recommend using a batch (`dataloader.bat`) file.

The application will return an exit code (refer to `echo %ERRORLEVEL%` in example command.

A batch or command script could appear as follows:

```
@echo off
eclipse\eclipsec -vm "%JRE_HOME%\bin" --launcher.suppressErrors -application
com.sap.custdev.projects.fbs.slc.dataloader.headless.dataloaderapplication
-noSplash
-console
-psConsoleLog -conDriver "D:\DBdrivers\Database_Drivers\mssql\sqljdbc4-4.0.jar"
-conName "con" -conPassword "password" -backendPassword "bkpassword"
-conFile "D:\DBdrivers\dataloader.xml"-createTables
-initialDownloadREM -Xdebug -Xnoagent -Djava.compiler=NONE-
Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005@echo EXIT CODE:
%ERRORLEVEL%
```

2.10.6.1.4 Exit Codes for Dataloader Execution

The exit codes for headless dataloader execution are given in the table below:

Exit Code	Description
0	All OK; Dataloader executed successfully.
303	Dataloader settings file not found. Please specify conFile.
304	Issues while loading dataloader settings.

Exit Code	Description
307	Database driver not found.
308	`backendPassword` not specified.
309	`conPassword` not specified

3 Solution Configuration Environment

Use

The solution configuration environment is part of the SAP Solution Sales Configuration application. It is used to define the configuration of a solution in a business document, such as a sales quotation or a sales order, and is integrated with sales order processing transactions in the following systems:

- SAP Customer Relationship Management (SAP CRM)
- SAP Enterprise Resource Planning (SAP ERP)
- Hybris Commerce Suite

Runtime Component

At run time, SAP Solution Sales Configuration comprises of the following components:

- A configuration UI: Launched from a sales application within the context of a sales document. It interacts with the engine via a command layer with well-documented APIs.
- A command layer: Exposes the functionality of the engine and content of a configuration session to the outside world.
- KBO - knowledge base orchestration layer: Enables multiple discrete knowledge bases to be accessed dynamically as required during a configuration session.
- SPE - Sales Pricing Engine: Accesses the pricing master data to calculate pricing.
- SCE - Sales Configuration Engine: Accesses knowledge base run time versions in the back-end database (if not already cached in memory). These can be solution knowledge bases generated from the solution modeling environment as well as product knowledge bases from CRM and ECC.

Components of the SCE:

Component	Description
PMS - pattern matching	Responsible for identifying the objects in the configuration that are "in scope", as defined by the objects/condition sections of constraints and rules. This process is called pattern matching. In addition, this component is responsible for forward chaining. That is, "firing" the constraints whose patterns are matched and altering the content of the configuration session, adding instances, or setting characteristic values.

Component	Description
TMS - truth maintenance	Tracks the facts asserted into the configuration session, along with their justifications. In addition, this component is responsible for retraction. That is, if a particular fact A is the justification for another fact B, and A is no longer true (for example, the user changes an entered value), the justification must be removed/retracted. Facts that are no longer justified must be removed from the configuration session. This may mean changing a characteristic value setting or deleting an instance.
DDB - dynamic database	This is the content of the configuration session. The PMS and TMS invoke methods on the `ddb_inst` class to request action in the configuration session, to set a value, or to create or remove an instance.

i Note

An event is logged for each action taken by these three components. The tracing and profiling capabilities of the engine support recording of these events, which are used in the solution modeling environment and, if enabled, in the production run time engine.

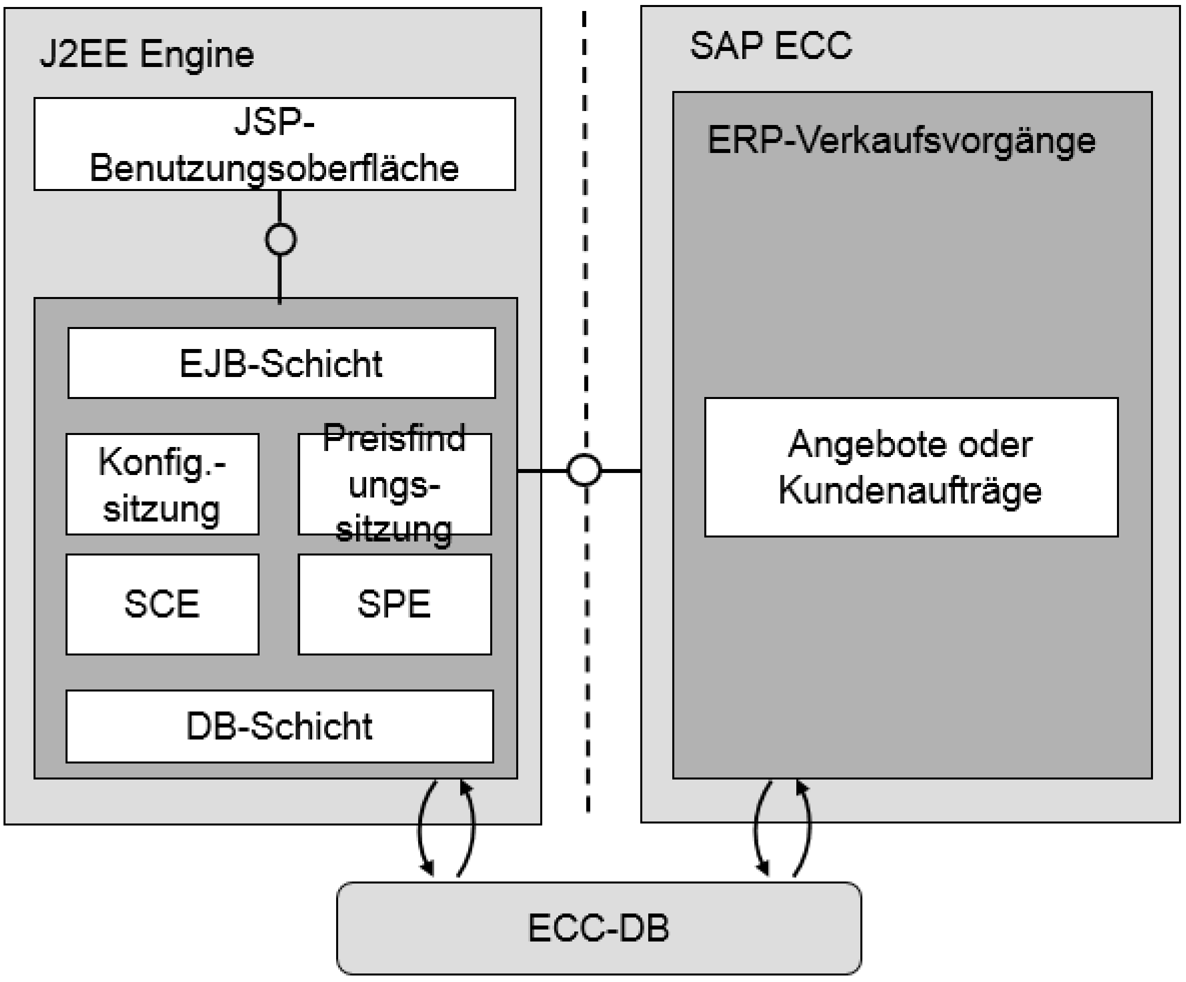
Integration

The solution configuration environment is built and packaged as an Enterprise Java Beans (EJB) component based on Java 2 Enterprise Edition (J2EE) standards. It is primarily intended to be deployed on an SAP J2EE NetWeaver Server.

The solution configuration environment includes the following components:

- Database (DB) Layer
- Sales Configuration Engine (SCE)
- Sales Pricing Engine (SPE)
- Configuration Session: This component encapsulates all of the configuration engine-related functionality and exposes the configuration engine commands to calling applications.
- Pricing Session: This component encapsulates all of the pricing-related functionality and exposes the pricing engine commands to calling applications.
- EJB Layer
- JavaServer Pages (JSP) User Interface (UI)

The following figure illustrates the architecture of the solution configuration environment when it is integrated with an SAP ERP system:



Architecture for Integrating Solution Configuration Environment with SAP ERP

The solution configuration environment for SAP Commerce is built and packaged as a Java Archive (.jar), which is added to the SAP Commerce class path. The Hybris service layer directly communicates with the Solution Configuration Engine (ConfigSession) "in-process", giving the best performance using Java native "call-by-reference".

The solution configuration environment integrates with the following business processes:

- Configure-to-Order in SAP CRM
- Configure-to-Order in Hybris
- Configure-to-Order in SAP ERP

When you choose to edit a configurable product in a sales document, the system opens the JSP UI. When you accept your configuration and close the JSP UI, you are returned to the SAP CRM or SAP ERP system, and the configuration result is reflected in the sales document, with any sales-relevant components listed as sub-line items. A saved configuration can be reopened, changed, and saved in the sales document, as necessary.

Features

- **Advanced Mode Configuration**
  SAP systems provide two ways of modeling solutions: compatible mode and advanced mode. Compatible mode is sufficient to model simple product configurations; however, advanced mode is required to model more complex solution configurations.
  SAP Internet Pricing and Configurator (IPC) is a component that is integrated into several standard SAP systems, including SAP CRM and SAP ECC. While the IPC can generate an advanced mode configuration, the configuration result is not integrated with the processes outside the IPC. Therefore, the advanced mode configuration result generated by the IPC cannot be used in the downstream standard business processes in the SAP CRM or SAP ERP systems. SAP Solution Sales Configuration overcomes this limitation by providing advanced mode configuration capabilities that are fully integrated with the standard configure-to-order business processes in SAP CRM and SAP ERP. For more information, see Advanced Mode Configuration [page 166].
- **Starting a Configuration from a Component**
  In addition to the top-down approach, where you add a solution to a sales document, and then configure the component parts of the solution, in SAP CRM, you can also begin a solution configuration by adding a component of the solution to a sales document. When you choose to configure the component, the system recognizes that it is part of a solution (based on your Customizing settings), and initializes the configuration using the knowledge base for the solution.
- **Complex Relationships Between Components**
  A bill of material (BOM) can be used to define simple relationships where one component comprises several subcomponents. SAP Solution Sales Configuration allows you to define more complex relationships between the components of a solution, for example, one-to-one and one-to-many relationships. These relationships are defined in the solution modeling environment as abstract data types (ADTs). Moreover, simple relationships between components can also be defined using ADTs. Therefore, SAP Solution Sales Configuration eliminates the need to maintain BOMs. The configuration engine interprets the ADTs and reflects them in the sales document. For more information, see One-to-Many Relationships Between Components [page 151].
- **Reference Characteristics**
  The system allows you to define reference characteristics as part of the solution model. The value of a reference characteristic can be used in the following ways:
  - Passed to the configuration as a context
  - Passed from the configuration to the sales document in the target system
- **Hard and Soft Ties in Follow-Up Documents**
  When you create follow-up documents for a quotation in SAP CRM, you can split the components of the solution into separate documents. The system allows you to control which components can be split into separate sales documents and which must remain together. For more information, see Hard and Soft Ties Between Components [page 152].
- **Interactive Pricing**
  During the solution configuration process, the pricing engine recalculates the total price of the solution after each change to the configuration. The total price of the current configuration and the delta price for each characteristic value are displayed in the user interface. Interactive pricing is an option that can be enabled or disabled in the configuration user interface. For more information, see Interactive Pricing and Delta Pricing [page 166].
- **Enhanced Solution Configuration User Interface (JSP UI)**
  The JSP UI is designed as an accordion with collapsible layers. It features a *Configuration* layer that contains a layer for each component in the solution.

The following icons are used to indicate the status of each component and characteristic:

- A green square indicates that the configuration is complete.
- A yellow triangle indicates that the configuration is incomplete.
- A gray diamond indicates that the field cannot be edited.

You can enable or disable the display of these icons using the *Settings* option in the *Configuration* layer. For more information about the functions in the JSP UI, see Creating Solution Configurations [page 183].

Customizing the Solution Configuration Engine

As a system administrator, you can also customize the behavior of the Solution Configuration Engine, by modifying the engine parameters. For more information about working with engine parameters, please refer to SAP Note 2291607 (Engine Parameters for Hybris SSC, SAP Solution Sales Configuration, and Solution Modeling Environment).

Constraints

The following features are available in SAP CRM, but not in SAP ERP or the Hybris Commerce Suite:

- Copying a solution configuration from one sales document to another
- Splitting the line items in a solution configuration into separate follow-up documents
- Starting a solution configuration from a component
- Copying a solution from one line item to another line item in the same sales document

→ Recommendation

If you want to use configuration mode B (Copy Configuration from Source Item) in Customizing for Copy Control, do not set the *Reexplode Structure/Free Goods* flag. If this flag is set, all the solution configuration data may not be copied.

With the *Reexplode Structure/Free Goods* flag not set, if you copy a line item within the same sales document, you must reopen the copied item's configuration and accept it to obtain the sub-items.

Related Information

Configure-To-Order in SAP CRM [page 150]
Configure-To-Order in Hybris [page 163]
Configure-To-Order in SAP ERP [page 156]

3.1 Configure-To-Order in SAP CRM

Use

You use this process to add a solution, or a component of a solution to a sales document, and then configure it in the standard SAP Customer Relationship Management (SAP CRM) WebClient user interface.

> i Note
>
> If you have installed SAP Solution Sales Configuration, the system displays the SAP Solution Sales Configuration JavaServer Pages user interface (JSP UI) for the configuration of both compatible mode and advanced mode products.

Prerequisites

- You have entered product master data and configuration master data.
- You have maintained the solution object to be used for starting the configuration of a component in Customizing for *Customer Relationship Management* under *Basic Functions* > *Solution Sales Configuration* > *Maintain Solution Objects for Products*.

Process

1. You create a sales document such as a package quotation, in the standard SAP CRM WebClient UI.
2. You add a solution, or a component of a solution, to the sales document.
   The system displays the solution or component, as a line item in the sales document.
3. You choose the *Edit* icon next to the line item for the solution or component.
   The system displays the configuration user interface for the solution and all of its component parts.

   > i Note
   >
   > If you have added a component of a solution to the sales document, the system initializes the configuration using the knowledge base for the solution. The configuration user interface displays the solution at the top level and the entered component (along with other components, if any) underneath.
   >
   > Note that the knowledge base for the solution must be modeled with the component (material) as one of the non-part instances.
4. You configure the solution.
5. You accept the configuration.
   The system closes the configuration user interface and displays the sales document. Each sales-relevant component of the solution is displayed as a line item.

i Note

You can also copy a solution configuration to a new sales document using the standard SAP CRM Online copy function.

You cannot copy a solution from one line item to another line item in a sales document.

More Information

- For information about the relationships between components, see One-to-Many Relationships Between Components [page 151] and Hard and Soft Ties Between Components [page 152].
- For information about splitting line items when creating follow-up documents, see Splitting Line Items in Follow-Up Document Creation [page 153].
- For information about the functions in the configuration user interface, see Creating Solution Configurations [page 183].

3.1.1 One-to-Many Relationships Between Components

Use

Solutions can include a mixture of interrelated components such as hardware components, software components, and services. There can be different kinds of relationships between these components, for example, has part, is part of, is served by, and so on. The system uses a special type of characteristic called an abstract data type (ADT) to model such relationships between components. ADT characteristics are defined in the solution modeling environment.

ADTs can be used to model one-to-many relationships. For example, a single service product can be associated with multiple hardware and software components.

Prerequisites

- You have modeled ADT characteristics for the solution in the solution modeling environment.
- You have maintained the names of the ADT characteristics to be used for different relationship types in Customizing for *Customer Relationship Management* under ▶ *Basic Functions* ▶ *Solution Sales Configuration* ▶ *Maintain Item Relationship Types* ◀.

More Information

Although you can model your own custom relationship types and the associated ADT characteristics, SAP Solution Sales Configuration provides the following two predefined relationship types:

- SALES_HARD (hard tie)
- SALES_SOFT (soft tie)

For more information about these predefined relationship types, see Hard and Soft Ties Between Components [page 152].

3.1.2 Hard and Soft Ties Between Components

Use

SAP Solution Sales Configuration provides the following predefined sales abstract data types (ADTs) to represent parent-child relationships between sales-relevant components:

- SALES_HARD
  Denotes a hard tie between two components
- SALES_SOFT
  Denotes a soft tie between two components

When you create follow-up documents for a solution quotation, you may want to split the components of the solution into separate follow-up documents. For example, you could have a package quotation for a solution that consists of a few hardware items and a few service items. You could create a sales order for the hardware items and a service contract for the service items.

However, certain components of the solution may be so closely related that they must be processed together. You control which components must be processed together by using the correct tie type to establish a relationship between those components. Components that are related by a hard tie must be processed as a group. They must remain together in the initial sales document and cannot be split into separate follow-up documents. Components with soft ties can be split into separate follow-up documents.

Prerequisites

- You have modeled sales ADT characteristics for the solution in the solution modeling environment.
- You have maintained the names of the hard and soft tie sales ADTs in Customizing for *Customer Relationship Management* under *Basic Functions* > *Solution Sales Configuration* > *Maintain Item Relationship Types*.

More Information

For more information, see Splitting Line Items in Follow-Up Document Creation [page 153].

3.1.3 Splitting Line Items in Follow-Up Document Creation

Use

You use this procedure to split line items in a package quotation into separate follow-up documents.

Prerequisites

- You have modeled sales abstract data type (ADT) characteristics for the solution in the solution modeling environment.
- You have maintained the names of the hard and soft-tying sales ADTs in Customizing for *Customer Relationship Management* under *Basic Functions* > *Solution Sales Configuration* > *Maintain Item Relationship Types*.
- You have maintained the copying control settings in Customizing for *Customer Relationship Management* under *Transactions* > *Basic Settings* > *Copying Control for Business Transactions*.
- You have created a package quotation for a solution.

Procedure

1. Open the package quotation.
2. Choose *Create Follow-Up*.
   The system displays a list of follow-up transactions.
3. Select the transaction type.
   The system displays a list of the line items that have either no ties or soft ties with their parents. Items that have hard ties with their parents are not shown in the list. The items are colored as follows:
   - Green denotes that the item can be moved to the follow-up document.
   - Red denotes that the item cannot be moved to the follow-up document.
4. Select the items to transfer to the follow-up document.
   You can freely select any items that have soft ties or no ties to other components. If you select (or deselect) an item that has a hard tie to another component, the system automatically selects (or deselects) its hard-tied components.
   If all of the components in the solution are selected, the solution is referred to as a 'full solution copy'.
   If only some of the components are selected, the solution is a 'trimmed (or partial) solution'.
   You can also deselect the highest level line item (solution item) and still select its components. This produces follow-up documents without the solution line item and without the solution level configuration.

Instead, each component item exists as a discrete high-level line item with its own discrete configuration result (assuming it is a configurable item). This action is referred to as 'stripping the solution'.

> i Note
>
> A special modeling technique is required to enable trimmed solutions. For more information, see SAP Note 2086153 (SSC Modeling for Trimmed Solutions).

5. Choose *Choose*.
   The system displays the follow-up document.

You can subsequently reconfigure the follow-up documents on an individual basis as you require.

More Information

- For more information, see **Hard and Soft Ties Between Components**
- > i Note
  >
  > While creating a follow-up document with configurable sub items, some characteristics' values may not be copied to the configuration of these items. If you observe this issue, please refer to SAP Note 2314099 (CRM: Utility to switch characteristics author during split).
- With SAP Solution Sales Configuration 3.0, splitting is now available for provider orders and provider contracts, in addition to sales and service transactions.

Related Information

Hard and Soft Ties Between Components [page 152]

3.1.4 Replicating Data from SAP CRM to SAP ERP

Use

You can configure a solution on the CRM user interface and replicate the data to the SAP ERP central component, where you can process it further. Replicating the data to SAP ERP means that you can take advantage of the functions and features offered by SAP ERP, such as creating a production order.

Prerequisites

- You have established an RFC connection between SAP CRM and SAP ERP.

- You have exported the solution models (knowledge bases) manually or via "headless export", from the solution modeling environment to SAP CRM and SAP ERP.

  You can find more information about this on the SAP Help Portal and navigate to the *SAP Solution Sales Configuration* > *Version 3.0* > *Installation Guide*.
- You have set up the necessary data structures and bills of material in SAP Solution Sales Configuration and SAP ERP.

Procedure

SAP Solution Sales Configuration enables you to create different types of documents in SAP CRM, such as sales quotations, service quotations, and contracts. The follow-up documents that are created in SAP CRM and then replicated to SAP ERP differ depending on the type of document originally created. For example, you can create a package quotation and then generate a service order that is replicated to SAP ERP. Alternatively, you can choose to create a sales order that is replicated to SAP ERP. In the steps below, we use the example of a sales order that is used to generate a production order in SAP ERP.

Create Quotation

1. In the SAP CRM Web client UI, create a package quotation (document type `SRVP`) and enter the business partners and accounts.
2. In the *Items* assignment block, enter the name of the product that you want to configure.
3. Choose the edit icon in the *Actions* column.
   The item details screen appears.
4. In the *Configuration* assignment block, specify the configuration elements that you require.
   Mandatory fields are indicated by a yellow triangle.
5. Choose the *Back* pushbutton to return to the quotation screen and then save.
   The system generates a document number for the quotation.

Create Sales Order

1. On the *Package Quotation* screen, choose the *Create Follow-Up* button and select the “sales process” document type.
2. In the pop up that appears, select the product that you configured in “Create Quotation” above.
3. Enter any missing information and save the sales order.
   The system generates a document number for the sales order and automatically replicates the sales order to SAP ERP.

Display Sales Order

You can display the sales order in SAP ERP to check that the data has been replicated correctly. To do so, proceed as follows:

1. In SAP ERP, call transaction `VA03` (*Display Sales Order*).
2. Enter the sales order number created in “Create Sales Order” above and press `ENTER`.
3. To check the replicated item data, select the material and choose the *Item details: Configuration* pushbutton.

i Note

If any of the replicated data is incorrect, you can change it by calling transaction `VA02` (*Change Sales Order*).

Create Production Order

1. In SAP ERP, call transaction `CO08` (*Create Production Order*).
   The sales order, item number, and production plant are entered by default. If this is not the case, enter them manually.
2. Enter the material that you configured earlier and press `ENTER`.
3. Choose the *Component Overview* pushbutton (`F6`) and generate a routing operation.
   The system shows an exploded view of the configurable material.
4. Save your data to create the production order.
5. Optional: Check your production order in the stock/requirements list by calling transaction `MD04` and entering the material. Your production order is listed in the *MRP element data* column.

3.2 Configure-To-Order in SAP ERP

Use

You use this process to add a solution to a sales document and configure it in the SAP Enterprise Resource Planning (SAP ERP) system.

> i Note
>
> SAP ERP has an integrated configuration engine called the Variant Configurator, which supports only compatible-mode configurations, however. If you install SAP Solution Sales Configuration, the SAP ERP system opens the SAP Solution Sales Configuration JavaServer Pages (JSP) user interface (UI) instead of the standard Variant Configurator user interface when you choose to configure a product (a solution) during the sales ordering process.
>
> An alternative to the Variant Configurator for compatible-mode configurations is the use of the IPC.
>
> A Customizing is available to select the VC or SSC configurator, based on transactions. You may customize entries in the table `/SLCE/DEL_TRAN`, to change the configurator if required. For more information about this, refer to SAP notes 2701187 (Configuration is not displayed for planned order and 2705646 (Configuration is not displayed for planned order - 2).

Prerequisites

- You have entered product master data and configuration master data
- You have marked the solution product as 'to-be-configured' with the IPC in the PME-VC (Product Modeling Environment-VC)

Process

1. You create a sales document, such as a quotation.
2. You add a solution to the sales document.
   The system displays the solution as a line item in the sales document.
3. You choose to configure the solution.
   The system displays the configuration user interface for the solution and all of its component parts.
4. You configure the solution.
5. You accept the configuration.
   The system closes the configuration user interface and displays the sales document. Each sales-relevant component of the solution is displayed as a line item.

More Information

For more information about the functions in the configuration user interface, see Creating Solution Configurations [page 183].

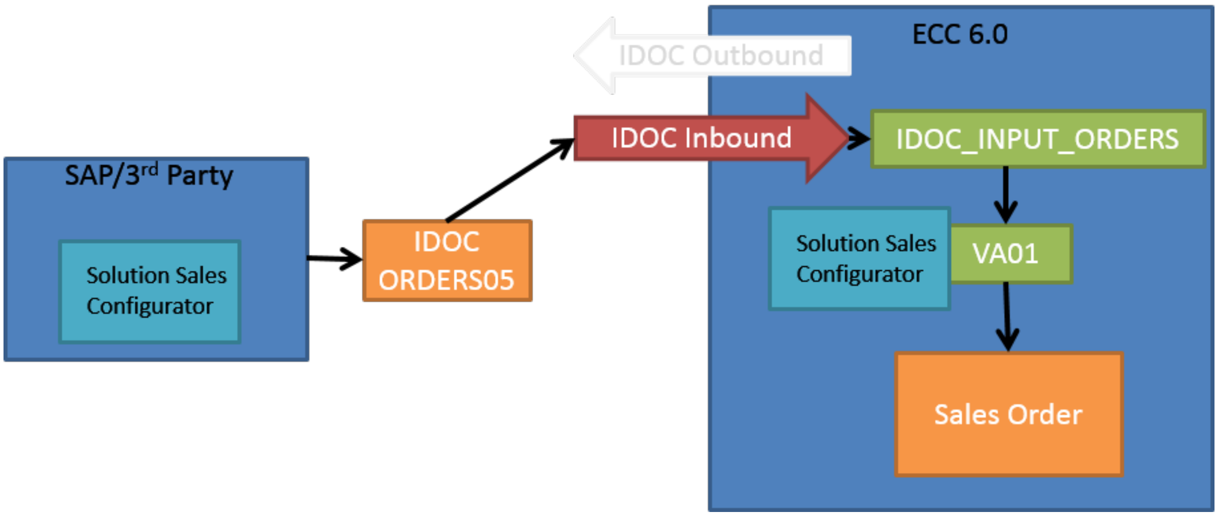
i Note

In SAP ERP, it is not possible to copy a solution configuration from one sales document to another or to split the line items for a solution into separate follow-up documents. It is also not possible to copy a solution from one line item to another line item in the same sales document.

3.2.1 IDOC Inbound Interface for Sales Order Creation

The business scenario for enhancement of ERP IDOC inbound interface for sales order creation has the following steps:

1. You send `IDOC`s from a source to a target system where the source system may be an SAP- or third-party system with SSC add-on installed and the target system is an SAP ERP system with the SSC add-on installed.
2. You use `ORDERS IDOC` inbound interface in the target ERP system to create sales documents with the configured items.
3. You use basic `IDOC` type `ORDERS05` and the function module for `IDOC_INPUT_ORDERS` standard inbound for `IDOC` processing.



IDOC Processing

An ERP solution configuration created by SSC operates in the following two modes at the same time:

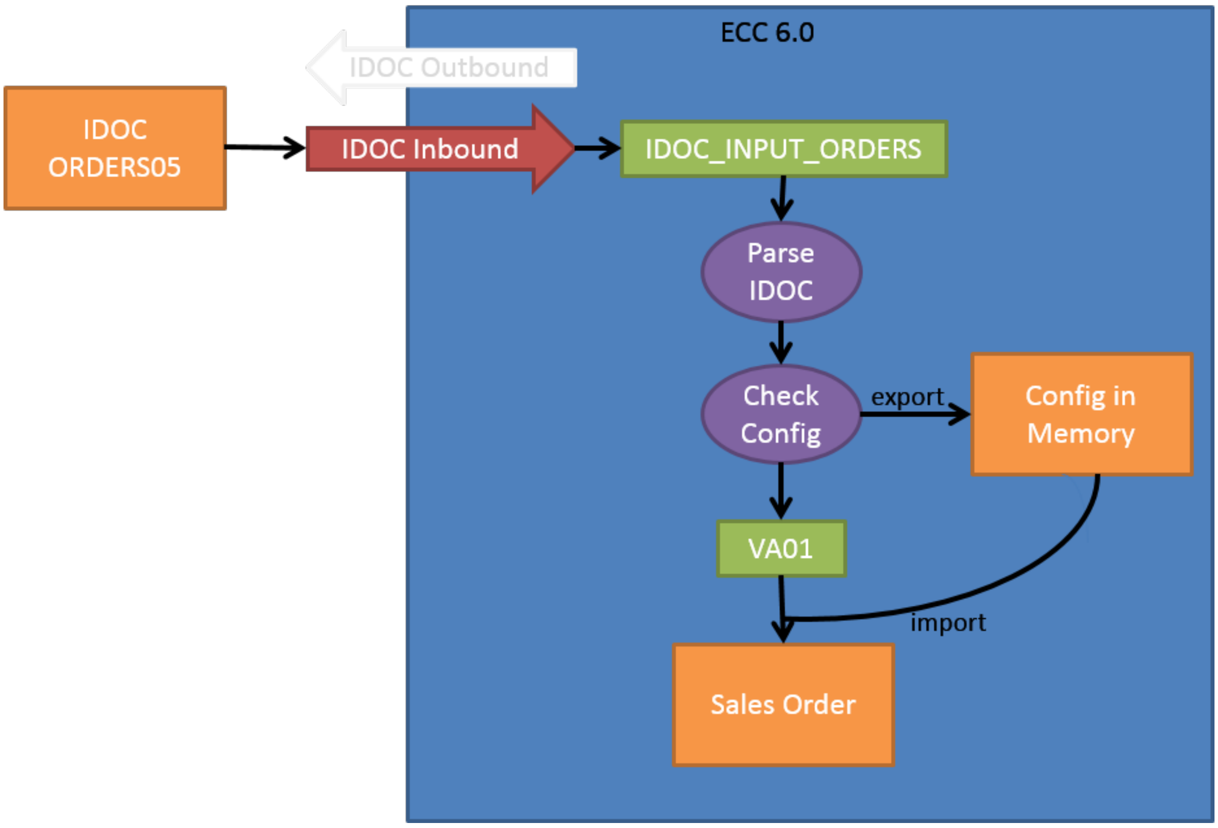
- The **Advanced Mode** model that contains the complete solution configuration, for example, including ADT characteristics
- The **Classic Mode** model, also known as the **Passive Receiving Structure (PRS)**, that contains only those parts of the configuration that are relevant for low-level production processes in SAP ERP. The PRS is returned and stored by SSC in the `CBase` format.
  The results of the advanced mode configuration are returned and stored by SSC in the rich-config XML format.

Basic `IDOC` type `ORDERS05` provides a set of segments to transfer configuration results, such as, `E1CUCFG`, `E1CUINS`, etc. These segments must be used to transfer the PRS. As of today, the rich-config XML cannot be transferred by a standard IDOC type or BAPI.

If you create sales orders by IDOCs that contain only the PRS configuration results but not the rich-config XML, then the users cannot see the full advanced mode configuration when they try to open the configuration manually in that sales order.

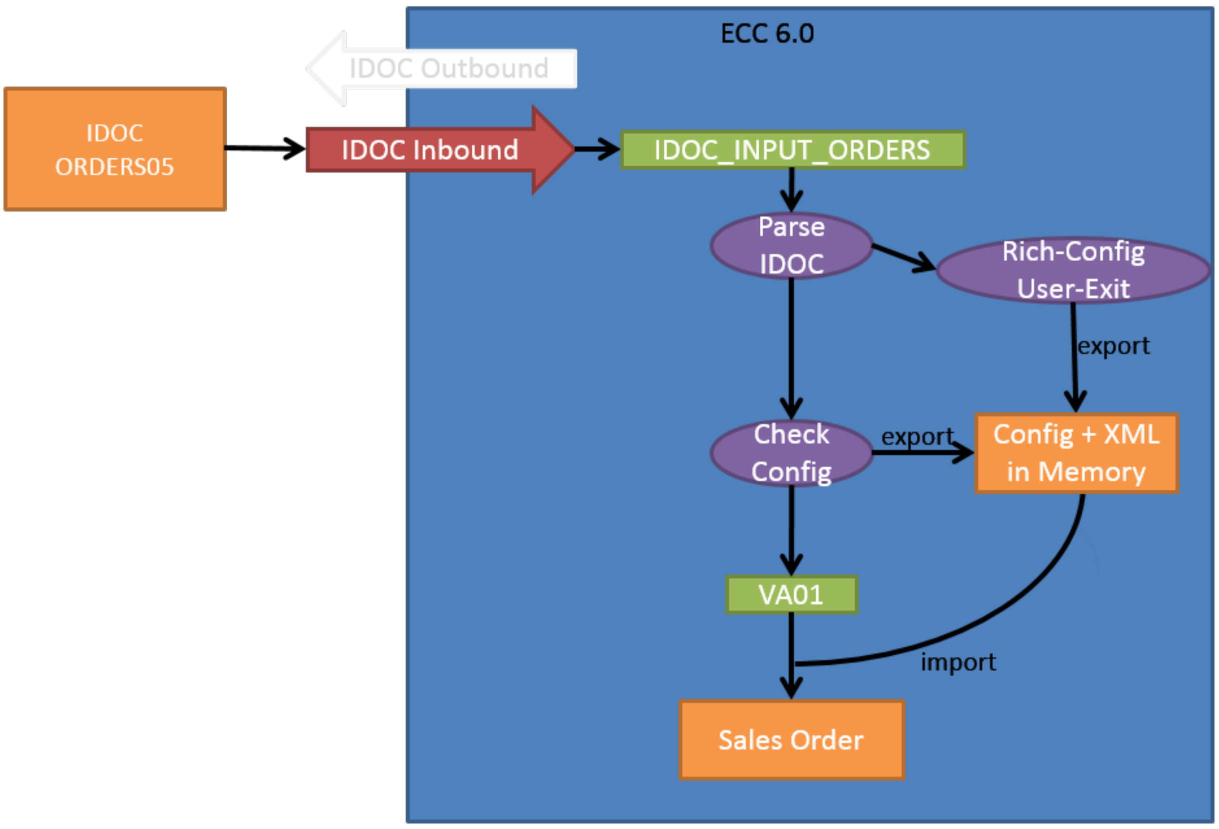
To solve this problem, SAP Notes 1996874 (Orders IDOC Inbound for SSC - config XML extension) and 2003665 (Orders IDOC Inbound for SSC - config XML extension (II)) enhance IDOC inbound processing in the following way:

- Enhancement of inbound function module `IDOC_INPUT_ORDERS`:
  - New user-exit that allows customers to read rich-config XML from own IDOC segment(s) or fetch the XML from an external source, for example, by an own web service
  - Functionality to export the retrieved XML(s) to ABAP memory
- Enhancement of sales document creation process:
  - If an XML is found in ABAP memory, it will be imported from memory
  - The XML will be assigned to the corresponding sales item



Before Implementation of Solution

After implementation of solution (Read: SAP Note 1996874 (Orders IDOC Inbound for SSC - config XML extension)).



After Implementation of Solution

You can use this process to create a sales document with items configured by IDOC. You will be able to view the advanced mode model by opening the configuration in SSC UI.

For more information about creating a sales order, refer to **Enhancement of ERP IDOC Inbound Interface for Sales Order Creation**.

Related Information

Enhancement of ERP IDOC Inbound Interface for Sales Order Creation [page 160]

3.2.1.1 Enhancement of ERP IDOC Inbound Interface for Sales Order Creation

i Note

You need to use the enhancements provided in SAP Notes 1996874 (Orders IDOC Inbound for SSC - config XML extension) and 2003665 (Orders IDOC Inbound for SSC - config XML extension (II)).

Prerequisites

The following prerequisites should be met for creation of a sales order:

- FBS Solution Sales Configurator (`SSC`) add-on for `SAP ERP` (`SLCE`) has been installed with the correct version and latest support package.
- SAP Note 2003665 (Orders IDOC Inbound for SSC - config XML extension (II)) has been implemented, including pre- and post-installation steps.
- The following notes need to be implemented as well:
  - SAP Note 1991156 (IDoc order creation: Sub-item without configuration tie)
  - SAP Note 1996874 (Order IDOC Inbound for SSC - config XML extension)
  - SAP Note 2006212 (Additional enhancement point in function group V45CU)
  - SAP Note 1923474 (Items on the Sales Document are deleted and recreated.)
  - SAP Note 1880466 (ECC:Create With Reference results in Exception on Config UI)
- If you have implemented and activated *BAdI /SLCE/CONFIGURE_BACKGROUND*, apply SAP Note 2455331 (ECC: Segregate method IS_ITEM_PROMOTION_ACTIVE from BAdI /SLCE/CONFIGURE_BACKGROUND) as well.

Assumptions

The following assumptions are considered during the process of enhancement:

- You have already customized `ORDERS IDOC` inbound interface in the target ERP system and are able to create sales documents by basic `IDOC` type `ORDERS05`.
- The sales documents are already created with the correct PRS configuration and only the rich-config XML is missing

Out of Scope

The out of scope processes and functions are especially but not limited to the following:

- Transfer and storage of custom relations in table /SLCE/IRT
- IDOC outbound interface
- IDOC interface for order change
- Other IDOC interfaces or BAPIs

Related Information

IDOC Inbound Interface for Sales Order Creation [page 157]

3.2.2 Light Engineer-to-Order

Use

Light engineer-to-order ("light ETO") enables you to split a complex configuration between the sales level and the engineering level, making the configuration process clearer. The processor of the production order BOM can also change the configuration on an individual basis in cases where a required change is too specific to be included in the configuration rules.

Process

1. The sales representative creates a sales order that contains a configurable material, either directly in SAP ERP (transaction `VA01`) or by replicating it from SAP CRM (see Replicating Data from SAP CRM to SAP ERP [page 154]).
2. The sales representative starts the sales configurator by choosing the *Item details: Configuration* pushbutton.
3. The sales configurator explodes the sales order BOM components.
4. The sales representative configures the components in the sales order BOM. The component may already be partly configured as the result of rules that have been defined in the system (see Defining Solution Dependencies [page 47]).

   i Note

   The component configured here is also the header of the production order BOM and is, therefore, visible in both the solution sales configurator and the variant configurator.
5. The sales representative accepts the configuration to transfer the result to the production order. The sales order is then saved and the sales order number communicated to the production engineer.
6. The production engineer opens the order BOM, for example, in transaction `CU51` by entering the order number and the item number that he or she wants to configure.

   i Note

   All standard engineering-to-order processes can be used (for example, transactions `CSKB`, `CS6x`, and `CU51`).

   The variant configurator opens and displays the header level of the engineering structure.
7. The production engineer navigates to the subitems and configures the relevant data.
8. The production engineer clicks the configuration tree structure to explode the BOM again. If there are any dependencies between the data (for example, the engineer has specified a component but not the component type), the system requests the missing data.
9. The production engineer saves the data.

Example

You have a material (`MY_SYSTEM`) that has been configured with the components `RACK`, `SUBRACK`, and `DEVICE`. The device slots into the subrack, which in turn slots into the rack.

The sales representative creates a sales order for the material `MY_SYSTEM` and starts the sales configurator. After the sales representative has specified the general data, the system displays three more dropdown areas - one for `RACK`, one for `SUBRACK`, and one for `DEVICE` (whereby `SUBRACK` and `DEVICE` are classes only). The sales representative configures the data for `RACK` and saves the order.

The production engineer opens the order BOM with the item number that is to be configured (in this example, item 20). The engineer navigates to `SUBRACK` in the configuration tree structure and enters the required data. He or she then explodes the BOM again and repeats the process to configure the required data for `DEVICE`.

3.2.2.1 Changing Bills of Material

Use

In certain cases, a production engineer may want to change the data in a bill of material for a specific production order. The required change is too specific to be included in the configuration rules, and so the engineer must change the data manually.

Procedure

1. Display the order BOM in transaction `CU51` and open the configuration result by choosing the *Result* pushbutton (`CTRL` + `F9`).
2. Select the checkbox for the component that you want to change and choose the *Item in Full* pushbutton (`SHIFT` + `F4`).
3. Change the required data, return to the configuration result, and save.

i Note

You can also add further items to the BOM by choosing the *Insert* pushbutton (`SHIFT` + `F1`). Examples of items that you can add include documents, texts, and compatible units.

3.3 Configure-To-Order in Hybris

Use

You use this process to configure a solution in Hybris and then add it to your shopping cart.

Prerequisites

- You have created product master data and configuration master data.
- You have created the solution product in your Hybris catalog.
- You have downloaded the master data from the SAP back-end system to the Hybris database using the Data Loader.
  For more information on using the Data Loader in Hybris, see the Hybris help portal and *Integrations and Data Management* > *SAP Integrations* > *SAP Product Configuration (On-Premise Edition)* > *Installing Product Configuration* > *Post Installation Steps* > *Configuring and Running the Data Loader* > *Loading Configuration Master Data through Data Loader*.

Process

1. You search for the solution product you require in the catalog.
2. You choose to configure the solution.
   The system displays the configuration user interface.
3. You configure the solution.
4. You add the solution to your shopping cart.
   The system closes the configuration user interface and displays your shopping cart.
5. You start the checkout process.

More Information

For more information about the functions in the configuration user interface, see Creating Solution Configurations [page 183].

3.3.1 Adding Related Products to the Solution

Context

SAP Solution Sales Configuration enables you to manually add related products to your solution.

Procedure

1. Open the *Find Related Products* dialog box.

   The system displays a list of related products. The default implementation of the search filter for “related products” uses the potential “non-part instances” of the solution model. You can enhance this with your own logic.
2. Select the product that you want to add to the solution.

   The system reports that the product has been added.
3. Continue to add further products or return to the solution.

Results

The system shows the added product in the section *Selected Products Related to This Solution*.

3.4 Component Promotion

Use

Component promotion (or “order stripping”) enables you to use the sales configurator to select any number of components, which you can then save to a sales order without the organizational structure (that is, the top node in the product or solution hierarchy).

For example, your company produces assembly lines, and you want to configure the entire assembly line. However, you do not manufacture the entire assembly line in your factory. Instead, you manufacture the machines that are sent to your customer, and the assembly line is then put together by the customer on site. In this case, you want to include the components of the assembly line in your sales order but not the assembly line itself.

Activities

The configuration modeler activates component promotion in the model by adding the reserved *Promote Subitems?* characteristic (`SSC_PROMOTE_SUBITEMS`) to the *Configuration Root* class and setting it to invisible with an assigned value of *Yes*.

If a user selects the value of this cstic as 'Y' during configuration, the root materials will be stripped, and all dependent sub-items will be promoted as root items.

i Note

Component promotion is inactive by default from SAP Solution Sales Configuration Version 2.0 Support Package 6 onwards. To enable it, follow this procedure:

If the item promotion is required, a new implementation can be created for the BAdI `/SLCE/ITEM_PROMOTION_SWITCH` and the logic provided with template implementation `/SLCE/EHI_SWITCH_ITEM_PROM` can be used to switch on the item promotion.

3.5 Advanced Mode Configuration

Use

SAP systems provide two modes of configuration: compatible mode and advanced mode. Compatible mode is suitable for simple product configurations and uses a super bill of material (super BOM). Advanced mode provides the additional flexibility required to configure complex solutions. It uses a dynamic BOM and abstract data types (ADTs).

Features

Configuring Complex Relationships

Advanced mode uses ADTs to define relationships outside those defined in the BOM. This means that more complex relationships can be defined, including one-to-one and one-to-many relationships. For example, a solution configuration could have a service item that relates to multiple hardware items.

You can display the ADTs by choosing *Show ADT Nodes* from the context menu and determine how each ADT is used.

Instantiating Multiple Subcomponents

Advanced mode uses a dynamic BOM, instead of the traditional super BOM used in compatible mode configurations. A dynamic BOM allows the instantiation of multiple subcomponents at the same BOM position.

3.6 Interactive Pricing and Delta Pricing

Use

During the solution configuration process, the system calculates and displays the price of the solution after each change in the configuration, for example, when the value of a characteristic is changed, when an instance is deleted, and when an instance is added. This is known as interactive pricing.

To inform the user about the impact of a characteristic selection on the total price, the system also displays the surcharge price or the price reduction next to each characteristic value in the configuration user interface. This is known as delta pricing.

Integration

The solution configuration process uses configuration and pricing data from the source system (SAP CRM or SAP ERP). To access the data, therefore, the system must be connected to the relevant database. This connection is implemented as a static destination setting between the J2EE environment, where the configuration engine is deployed, and the source system (SAP CRM or SAP ECC).

To optimize performance, pricing and configuration data is cached at Java stack level.

Prerequisites

- You have maintained prices (list prices) for the required products.
- You have maintained one-to-one variant conditions in the solution configuration model to reflect the surcharge or the price reduction for each characteristic value.

> → Recommendation
>
> To optimize system performance, SAP recommends that you maintain a pricing procedure that is dedicated to interactive pricing purposes. This pricing procedure should hold only the necessary condition types and condition tables relevant for interactive pricing.

Activities

Interactive Pricing

To calculate and display interactive prices, the system performs the following steps:

1. The order mapper interprets the solution configuration and converts it into a sales product structure that can be used by the sales pricing engine (SPE).
   The sales product structure is represented in the configuration result by the following types of instance relationship:
   - Sales abstract data type (ADT) characteristics (see Characteristic [page 24])
   - Compatible mode sales bill of material (BOM) explosions of the super BOM
2. The system passes the sales product structure to the SPE.
3. The system triggers the pricing determination process.
   The SPE calculates the total price of the solution, which is a summation of all the instance (product) list prices and all the variant conditions attached to the selected characteristic values.
4. The system passes the result of the pricing determination process to the configuration user interface and updates the display.

⚠ Caution

The pricing procedure used for interactive pricing should not propagate the prices of subitems to the root item price.

Delta Pricing

To calculate and display delta prices, the system performs the following steps:

1. The surcharge or reduction price for each pricing-relevant characteristic value is retrieved and displayed next to each characteristic value in the user interface.
2. After each change to the configuration, the surcharge or reduction price is recalculated based on the user's last selection.
   It is assumed that a one-to-one characteristic value/variant condition assignment is used. Therefore, the new delta prices only need to be calculated for the values of the characteristic that was changed last.
   The following formula is used to calculate the delta price:
   characteristic value price = variant condition value price - selected characteristic value price
   If the value is positive, the delta price is a surcharge. If the value is negative, the delta price is a reduction.

More Information

i Note

You can enable and disable interactive pricing and delta pricing using a checkbox in the configuration user interface.

For more information about using the configuration user interface, see Creating Solution Configurations [page 183].

3.6.1 Pricing Formula and User Exits

Pricing is a highly customizable and configurable engine. However, in some cases, the regular features and functionalities provided by the pricing engine are not sufficient. In such cases, it is possible to meet the special business requirements by using **Pricing Formulas** and **User Exits**. These are custom functions that allow customization of the default behavior of existing pricing conditions.

For more information about this, refer to the SAP Help Portal and navigate to *Basic Functions and Master Data in SD Processing (SD-BF)* › *Basic Functions in SD* › *Pricing and Conditions*.

3.6.1.1 Available User Exits and APIs

Here, you can find all the relevant information related to the available pricing-related user-exit types. First, the standard features are explained and then the different types of user exits. The parameters that form the interface between pricing and user exits are also described briefly.

3.6.1.1.1 Logging Capabilities

For customer pricing user exits, there is an easy way to include fast logging. The `com.sap.spe.base.logging.UserexitLogger` class implements two methods for logging debug messages or error messages. Logging is fast and done only if the appropriate log level is reached, which you can define at runtime.

ZSpecialRoundingValueFormula (shorten)

```
package your.company.pricing.userexits;

import com.sap.spe.base.logging.UserexitLogger;
[..]
public class ZSpecialRoundingValueFormula extends ValueFormulaAdapter {

    private static UserexitLogger userexitlogger =
        new UserexitLogger(ZSpecialRoundingValueFormula.class);

    public BigDecimal overwriteConditionValue(IPricingItemUserExit item,
            IPricingConditionUserExit condition) {
[..]
        userexitlogger.writeLogDebug("old cond value: "
                + val.getValueAsString());
[..]
    }
}
```

Line No.	Description
8	Create a static instance of the `UserexitLogger` class. As constructor parameter, pass the actual class.
13	Use `writeLogDebug(String s)` or `writeLogError(String s)` to log the string s in the log.

3.6.1.1.2 Condition Base Formula

The condition base formula can be used to overturn the automatically calculated base value of a condition. This type of user exit must be assigned in Customizing to the user exits type `BAS` (condition base formula).

This user exit is called after the condition base value has been calculated for each pricing condition. The user exit class must be inherited from `BaseFormulaAdapter` and implement method overwriteConditionBase. The overwriteConditionBase method has the parameters pricingItem and pricingCondition, which represents the item and the actual condition.

If this method returns a null object reference, pricing will keep the base value that is called automatically.

ZSpecialBaseFormula

```
package your.company.pricing.userexits;

import java.math.BigDecimal;

import com.sap.spe.base.logging.UserexitLogger;
import com.sap.spe.conversion.IDimensionalValue;
import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
import com.sap.spe.pricing.transactiondata.userexit.BaseFormulaAdapter;

public class ZSpecialBaseFormula extends BaseFormulaAdapter {

    private static UserexitLogger userexitlogger =
        new UserexitLogger(ZSpecialBaseFormula.class);

    public BigDecimal overwriteConditionBase(IPricingItemUserExit pricingItem,
            IPricingConditionUserExit pricingCondition) {

            BigDecimal result;

            userexitlogger.writeLogDebug("old cond base: "
                    + pricingCondition.getConditionBase().getValueAsString());

            // double the base value
            result = pricingCondition.getConditionBase().getValue().
                         multiply(new BigDecimal("2"));

            userexitlogger.writeLogDebug("new cond base: " + result);

            return result;
        }
}
```

Line No.	Description
11	Extend/subclass the API BaseFormulaAdapter.
16	Overwrite the implementation of the overwriteConditionBase method.
25	Change the value of the automatically determined condition base.
30	Return the changed condition base value.

3.6.1.1.3 Item Calculation Begin Formula

This seldom-used user exit is available to change the document and item if necessary, before item pricing takes place. This type of user exit must be assigned in Customizing to user exit type `CAB` (Item Calculation Begin Formula).

The user exit class must be inherited from `PricingItemCalculateBeginFormulaAdapterprDocument`) and the item (. It passes a reference to the pricing document (`prItem`).

ZSpecialCalculationBeginFormula

```
package your.company.pricing.userexits;

import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
import
com.sap.spe.pricing.transactiondata.userexit.PricingItemCalculateBeginFormulaAdap
ter;

public class ZSpecialCalculationBeginFormula extends
PricingItemCalculateBeginFormulaAdapter {
{

    private int stepNumber, counter;

    public void calculationBegin(IPricingDocumentUserExit prDocument,
         IPricingItemUserExit prItem) {

    stepNumber = 10;
    counter = 1;

    prDocument.setZeroPriceActive(true);

    }
}
```

Line No.	Description
7	. It passes aExtend the API `PricingItemCalculateBeginFormulaAdapter`
12	Overwrite the implementation of the `calculationBegin` method
18	Set the document to accept zero prices as valid prices

3.6.1.1.4 Item Calculation End Formula

This seldom-used user exit is available to change the document and item if necessary, after item pricing has taken place. This type of user exit must be assigned in Customizing to user exit type `CAE` (Item Calculation End Formula).

The user exit class must be inherited from `PricingItemCalculateEndFormulaAdapter`. It passes a reference to the pricing document (`prDocument`) and the item (`prItem`).

ZSpecialCalculationEndFormula

```
package your.company.pricing.userexits;

import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
import
com.sap.spe.pricing.transactiondata.userexit.PricingItemCalculateEndFormulaAdapte
r;

public class ZSpecialCalculationEndFormula extends
PricingItemCalculateEndFormulaAdapter {

    private int stepNumber, counter;

    public void calculationEnd(IPricingDocumentUserExit prDocument,
        IPricingItemUserExit prItem) {

    stepNumber = 10;
    counter = 1;
    prItem.findPricingCondition(stepNumber, counter).setConditionControl('A');

    }
}
```

Line No.	Description
7	`PricingItemCalculateEndFormulaAdapter`
11	Overwrite the implementation of the Extend the API `calculationEnd` method
16	Extend the APISet the condition control of the pricing condition at `stepNumber` 10 to Automatic A

3.6.1.1.5 Configuration Formula

This seldom-used user exit is called when the product configuration process creates subitems. This type of user exit must be assigned in Customizing to user exit type `CFG` (Configuration Formula), which is called for subitems created by SCE.

The user exit class must be inherited from `SPCSubItemCreatedByConfigurationFormulaAdapter`. For each subitem, method `isRelevantForPricing` is called and a reference to the new subitem and the configuration instance is passed.

ZSpecialConfigurationFormula

```
package your.company.pricing.userexits;

import com.sap.spc.document.userexit.ISPCItemUserExitAccess;
import com.sap.sce.front.base.Instance;
import
com.sap.spc.document.userexit.SPCSubItemCreatedByConfigurationFormulaAdapter;

public class ZSpecialConfigurationFormula extends
    SPCSubItemCreatedByConfigurationFormulaAdapter {

    public boolean isRelevantForPricing(ISPCItemUserExitAccess subItem,
            Instance instance) {

    return subItem.isRelevantForPricing();
    }
}
```

Line No.	Description
8	Extend the API `SPCSubItemCreatedByConfigurationFormulaAdapte r`
10	Implement the `isRelevantForPricing` method
13	Set the configuration subitem to pricing-relevant or not

3.6.1.1.6 Condition Init Formula

After a pricing condition has been initialized, it can be changed with this user exit, which is called whenever an internal condition (a transactional object or business entity) is created. This type of user exit must be assigned in Customizing to user exit type `CNI` (Condition Init Formula).

The user exit class must be inherited from `PricingConditionInitFormulaAdapter` and must overwrite the method `init`. It allows the new condition to be changed (parameter `prCondition`).

ZSpecialConditionInitFormula

```
package your.company.pricing.userexits;

import java.math.BigDecimal;

import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
import
com.sap.spe.pricing.transactiondata.userexit.PricingConditionInitFormulaAdapter;
```

```
public class ZSpecialConditionInitFormula extends
PricingConditionInitFormulaAdapter {

    public void init(IPricingDocumentUserExit prDocument, IPricingItemUserExit
prItem,
            IPricingConditionUserExit prCondition) {

    if (prCondition.getConditionTypeName() != "0PR0"
                     && prCondition.getChangeOfRateAllowed())
    prCondition.setConditionRateValue(new BigDecimal("2"));
    }
}
```

Line No.	Description
10	Extend the API `PricingConditionInitFormulaAdapter`
12	Implement the `init` method
17	Set the condition rate to 2

3.6.1.1.7 Copy Formula

While a document is being copied, the pricing condition can be fixed or other changes can take place if required. This type of user exit must be assigned in Customizing to user exit type `CPY` (Copy Formula).

This user exit is called during the copying process. The user exit class must be inherited from class `PricingCopyFormulaAdapter` and implement method `pricingCopy`. Parameters `pricingDocument`, `pricingItem`, and `pricingCondition` are references to the target document, item, and condition. The pricing type describes what should happen to the pricing result when new pricing takes place. The parameter `copyType` is a reference to the Customizing used for the copy process; `sourceSalesQuantity` contains the old quantity of the source item.

ZSpecialCopyFormula

```
package your.company.pricing.userexits;

import com.sap.spe.conversion.IQuantityValue;
import com.sap.spe.pricing.customizing.ICopyType;
import com.sap.spe.pricing.customizing.IPricingType;
import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
import com.sap.spe.pricing.transactiondata.userexit.PricingCopyFormulaAdapter;

public class ZSpecialCopyFormula extends PricingCopyFormulaAdapter {

    public void pricingCopy(IPricingDocumentUserExit pricingDocument,
IPricingItemUserExit
                            pricingItem, IPricingConditionUserExit
pricingCondition,
                            IPricingType pricingType, ICopyType copyType,
IQuantityValue
                            sourceSalesQuantity) {

      // fix condition value and base
      pricingCondition.setConditionControl('E');
        }
}
```

Line No.	Description
11	Extend the API `PricingCopyFormulaAdapter`
13	Overwrite the implementation of the `pricingCopy` method
19	Fix the conditions value and base by setting the `conditionControl` to E

3.6.1.1.8 Document Init Formula

After a pricing document has been initialized, it can be changed with this user exit, which is called when a new pricing document is created. This type of user exit must be assigned in Customizing to user exits type `DOI` (Document Init Formula).

The user exit class must be inherited from class `PricingDocumentInitFormulaAdapter` and implement method `init`. A reference to the new document is passed.

ZSpecialDocumentInitFormula

```
package your.company.pricing.userexits;

import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import
com.sap.spe.pricing.transactiondata.userexit.PricingDocumentInitFormulaAdapter;

public class ZSpecialDocumentInitFormula extends
PricingDocumentInitFormulaAdapter {

    public void init(IPricingDocumentUserExit prDocument) {
      if (!prDocument.isAlwaysPerformingGroupConditionProcessing())
        prDocument.setAlwaysPerformingGroupConditionProcessing(true);
    }
}
```

Line No.	Description
6	Extend the API `DocumentInitFormula`

Line No.	Description
8	Overwrite the implementation of the `init` method
10	Set group condition processing to active

3.6.1.1.9 Group Key Formula

This user exit can be used to replace the automatically determined condition value. This type of user exit must be assigned in Customizing to user exit type `VAL` (Condition Value Formula).

This user exit is called after the condition value has been calculated for each pricing condition.
The user exit class must be inherited from class `ValueFormulaAdapter` and implement at least `overwriteConditionValue`. If group condition processing is enabled for the condition type, the implementation of method `overwriteGroupConditionValue` is possible. Both methods can return null to indicate that the original value is to be taken.

ZSpecialRoundingValueFormula

```
package your.company.pricing.userexits;

import java.math.BigDecimal;

import com.sap.spe.base.logging.UserexitLogger;
import com.sap.spe.conversion.ICurrencyValue;
import com.sap.spe.pricing.transactiondata.userexit.IGroupConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
import com.sap.spe.pricing.transactiondata.userexit.ValueFormulaAdapter;

public class ZSpecialRoundingValueFormula extends ValueFormulaAdapter {

    private static UserexitLogger userexitlogger =
        new UserexitLogger(ZSpecialRoundingValueFormula.class);

    public BigDecimal overwriteConditionValue(IPricingItemUserExit item,
            IPricingConditionUserExit condition) {
        BigDecimal result;

        ICurrencyValue val = condition.getConditionValue();
        userexitlogger.writeLogDebug("old cond value: "
                + val.getValueAsString());

        result = val.getValue().setScale(0, BigDecimal.ROUND_HALF_UP);

        BigDecimal qnt = item.getProductQuantity().getValue();
        qnt = qnt.divide(new BigDecimal("100"), 2, BigDecimal.ROUND_HALF_UP);

        userexitlogger.writeLogDebug("new cond value: " + result.subtract(qnt));

        return result.subtract(qnt);
    }
    public BigDecimal overwriteGroupConditionValue(
            IPricingDocumentUserExit item, IGroupConditionUserExit condition) {
        // do nothing
        return null;                    }
}
```

Line No.	Description
13	Extend the API `ValueFormulaAdapter`
18	Overwrite the implementation of the `overwriteConditionValue` method
33	Change the value of the automatically determined condition value
33	Return the changed condition value
36	Overwrite the implementation of the `overwriteGroupConditionValue` method
39	Return null to keep the automatically calculated value

3.6.1.1.10 Item Init Formula

After the pricing item has been initialized, it can be changed with this user exit, which is called when a new pricing item is created. This type of user exit must be assigned in Customizing to user exit type `ITI` (Item Init Formula).

The user exit class must be inherited from class `PricingItemInitFormulaAdapter` and implement method `init`. A reference to the document and to the new item is passed.

ZSpecialItemInitFomula

```
package your.company.pricing.userexits;

import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
import
com.sap.spe.pricing.transactiondata.userexit.PricingItemInitFormulaAdapter;

public class ZSpecialItemInitFomula extends PricingItemInitFormulaAdapter {

    public void init(IPricingDocumentUserExit prDocument, IPricingItemUserExit
prItem) {
      if (prItem.isStatistical())
        prItem.setExclusionFlag('$');
    }
}
```

Line No.	Description
7	Extend the API `ZSpecialItemInitFomula`
9	Overwrite the implementation of the `init` method
11	Set the item exclusion flag

3.6.1.1.11 Pricing Init

In previous releases, this user exit was called `CRMDocumentStandardExit` where it was used mainly to pass header attributes to be used in method `initializeDocument`. As of Release 2.0 SP5, these attributes can be customized. Pricing Init user exits can now be used only to set the unit of rounding to the smallest unit of a currency. This type of user exit must be assigned in Customizing to the user exit type `PRI` (Pricing Init).

This user exit is called when a new pricing document is created. The user exit class must be inherited from class `PricingInitFormulaAdapter` and must implement method `initializeDocument`. This method has parameter `documentUserExitAccess`, which represents the pricing document.

ZPricingInit

```
package your.company.pricing.userexits;

import com.sap.spe.document.userexit.IDocumentUserExitAccess;
import com.sap.spe.document.userexit.PricingInitFormulaAdapter;

public class ZPricingInit extends PricingInitFormulaAdapter {

    public void initializeDocument(IDocumentUserExitAccess
documentUserExitAccess) {

      documentUserExitAccess.setUnitToBeRoundedTo(20);
    }
}
```

Line No.	Description
6	Extend the API `PricingInitFormulaAdapter`
8	Overwrite the implementation of the `initializeDocument` method
10	Set the rounding unit to 20

3.6.1.1.12 Pricing Prepare

In previous releases, this user exit was called `CRMItemStandardExit`. Pricing Prepare user exits can be used to add header and/or item attributes to be used during the pricing process. These attributes can now be customized. This type of user exit must be assigned in Customizing to the user exit type `PRP` (Pricing Prepare).

The Pricing Prepare user exit is called when creating a new pricing item and when new pricing takes place. The user exit class must be inherited from class `PricingPrepareFormulaAdapter` and must implement method `addAttributeBindings`. This method has parameter `itemUserExitAccess`, which represents the pricing item.

ZPricingPrepare

```
package your.company.pricing.userexits;

import com.sap.spe.document.userexit.IItemUserExitAccess;
import com.sap.spe.document.userexit.PricingPrepareFormulaAdapter;

public class ZPricingPrepare extends PricingPrepareFormulaAdapter {

  public void addAttributeBindings(IItemUserExitAccess itemUserExitAccess) {

    itemUserExitAccess.addAttributeBinding("ZLAND", "DE");
    }
}
```

Line No.	Description
6	Extend the API `PricingPrepareFormulaAdapter`
8	Overwrite the implementation of the `addAttributeBindings` method
10	Set the attribute `ZLAND` to the value “DE”.

3.6.1.1.13 Requirement

This user exit is used during condition determination at pricing procedure step/counter level and at condition access step level. This type of user exit must be assigned in Customizing to the user exit type `REQ` (Requirement).

The user exit class must be inherited from `RequirementAdapter` and implement method `checkRequirement`. If this method returns false, the actual access is not made.

ZSpecialRequirement

```
package your.company.pricing.userexits;

import com.sap.spe.base.logging.UserexitLogger;
import com.sap.spe.condmgnt.customizing.IAccess;
import com.sap.spe.condmgnt.customizing.IStep;
import com.sap.spe.condmgnt.finding.userexit.IConditionFindingManagerUserExit;
import com.sap.spe.condmgnt.finding.userexit.RequirementAdapter;

public class ZSpecialRequirement extends RequirementAdapter {

    private static UserexitLogger userexitlogger =
        new UserexitLogger(ZSpecialRequirement.class);

    public boolean checkRequirement(IConditionFindingManagerUserExit item,
        IStep step, IAccess access)
    {
        String zland = item.getAttributeValue("ZLAND");
        if (zland == null || zland.equals("")) {
          userexitlogger.writeLogError("ZLAND attribute missing");
             return false;
        } else {
             return zland.equals("US");
        }
     }
}
```

Line No.	Description
9	Extend the API `RequirementAdapter`
14	Overwrite the implementation of the `checkRequirement` method
17	Retrieve an attribute value to be used for the check
22	Return the check result: “true” to make the access, “false” not to make the access

3.6.1.1.14 Scale Base Formula

This user exit can be used to replace the automatically determined scale base. This type of user exit must be assigned in Customizing to the user exit type `SCL` (Scale Base Formula).

This user exit is called after the condition-scale base value has been calculated for a pricing condition. The user exit class must be inherited from class `ScaleBaseFormulaAdapter` and implement at least `overwriteScaleBase`. If group condition processing is enabled for the condition type, method `overwriteGroupScaleBase` can also be implemented. Both methods can return null to indicate that the original value is to be taken.

ZSpecialScaleBaseFormula

```
package your.company.pricing.userexits;

import java.math.BigDecimal;

import com.sap.spe.base.logging.UserexitLogger;
import com.sap.spe.pricing.transactiondata.userexit.IGroupConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
import com.sap.spe.pricing.transactiondata.userexit.ScaleBaseFormulaAdapter;

public class ZSpecialScaleBaseFormula extends ScaleBaseFormulaAdapter {

    private static UserexitLogger userexitlogger =
        new UserexitLogger(ZSpecialScaleBaseFormula.class);

    public BigDecimal overwriteScaleBase(IPricingItemUserExit item,
            IPricingConditionUserExit condition,
            IGroupConditionUserExit groupCondition) {

        userexitlogger.writeLogDebug("Old scale: " +
                       groupCondition.getConditionScale().getValueAsString());

        if (groupCondition.getConditionScale() != null) {
             return groupCondition.getConditionScale().getValue().setScale(0,
                           BigDecimal.ROUND_FLOOR);
        }
        else
        {
             return null;
        }
        }

        public BigDecimal overwriteGroupScaleBase(IPricingDocumentUserExit
document,
            IGroupConditionUserExit groupCondition) {

          if (groupCondition.getConditionScale() != null) {
              return groupCondition.getConditionScale().getValue().setScale(0,
                        BigDecimal.ROUND_FLOOR);
          }
          else
          {
              return null;
          }
        }
}
```

Line No.	Description
12	Extend the API `ScaleBaseFormulaAdapter`
17	Overwrite the implementation of the `overwriteScaleBase` method
25	Return the changed scale base value

Line No.	Description
34	Overwrite the implementation of the `overwriteGroupScaleBase` method
38	Return the changed scale base value

3.6.1.1.15 Condition Value Formula

This user exit can be used to replace the automatically determined condition value. This type of user exit must be assigned in Customizing to user exit type `VAL` (Condition Value Formula).

This user exit is called after the condition value has been calculated for each pricing condition.
The user exit class must be inherited from class `ValueFormulaAdapter` and implement at least `overwriteConditionValue`. If group condition processing is enabled for the condition type, the implementation of method `overwriteGroupConditionValue` is possible. Both methods can return null to indicate that the original value is to be taken.

ZSpecialRoundingValueFormula

```
package your.company.pricing.userexits;

import java.math.BigDecimal;

import com.sap.spe.base.logging.UserexitLogger;
import com.sap.spe.conversion.ICurrencyValue;
import com.sap.spe.pricing.transactiondata.userexit.IGroupConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingConditionUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingDocumentUserExit;
import com.sap.spe.pricing.transactiondata.userexit.IPricingItemUserExit;
import com.sap.spe.pricing.transactiondata.userexit.ValueFormulaAdapter;

public class ZSpecialRoundingValueFormula extends ValueFormulaAdapter {

    private static UserexitLogger userexitlogger =
        new UserexitLogger(ZSpecialRoundingValueFormula.class);

    public BigDecimal overwriteConditionValue(IPricingItemUserExit item,
            IPricingConditionUserExit condition) {
        BigDecimal result;

        ICurrencyValue val = condition.getConditionValue();
        userexitlogger.writeLogDebug("old cond value: "
                + val.getValueAsString());

        result = val.getValue().setScale(0, BigDecimal.ROUND_HALF_UP);

        BigDecimal qnt = item.getProductQuantity().getValue();
        qnt = qnt.divide(new BigDecimal("100"), 2, BigDecimal.ROUND_HALF_UP);

        userexitlogger.writeLogDebug("new cond value: " + result.subtract(qnt));

        return result.subtract(qnt);
    }

    public BigDecimal overwriteGroupConditionValue(
            IPricingDocumentUserExit item, IGroupConditionUserExit condition) {
        // do nothing
        return null;
    }
}
```

Line No.	Description
13	Extend the API `ValueFormulaAdapter`
18	Overwrite the implementation of the `overwriteConditionValue` method
33	Change the value of the automatically determined condition value
33	Return the changed condition value
36	Overwrite the implementation of the `overwriteGroupConditionValue` method
39	Return null to keep the automatically calculated value

3.6.2 Modifying Pricing Context

If you want to change the pricing context for the header or item of a document, you can implement the `/SLCC/MODIFY_PRICING_CONTEXT` BadI. SAP delivers a default implementation of this BadI and customers can have their own implementation as this is a multiple use BadI.

3.7 Creating Solution Configurations

Use

This procedure explains how to use the main functions in the advanced-mode configuration user interface. The user interface is presented as an accordion with collapsible layers. Within the *Configuration* layer, there is a layer for each component in the solution.

Prerequisites

You have created a sales document, added an advanced mode product, and chosen to edit it.

Procedure

To access individual functions shown in the table, use the *Configuration* layer of the user interface:

Function	Navigation	More Information
Add a non-part instance	Choose *Add Non Part Instance*	The system displays a list of all the non-part instances for the solution. You can add one or more non-part instances by choosing the *Add Component* pushbutton next to the relevant component.
Enable or Disable Interactive Pricing	Select or deselect the *Interactive Pricing* checkbox	The system enables or disables both interactive pricing (display of the total price) and delta pricing (display of surcharges and price reductions).
Display the sales structure for the solution	Select the *Sales Structure* checkbox	The system displays the solution configuration in one of the following ways: • **Flat Structure** This view displays the non-part instances in a list and the bill of material (BOM) explosion as a hierarchical structure. The relationships between the non-part instances are not displayed. • **Sales Structure** This view displays the hierarchical relationships between the instances, which are interpreted from the sales abstract data types (ADTs) and the BOM explosion.

Function	Navigation	More Information
View or specify the configuration settings	Choose *Settings*	The system displays the following information: • Extended Configuration Management (XCM) Application Configuration • Knowledge Base Description • Knowledge Base Version • Knowledge Base Profile • Knowledge Base Build Number Here, you can also enable/disable the following options: • Display Invisible Characteristics • Display Language-Dependent Descriptions • Show Characteristic Groups • Display All Options • Show Status Lights • Evaluate Characteristics Online • Render Values in Multiple Columns • Enable JQuery Controls • Show Assignable Values only • Indent Components • Select Price Type • Enable Customization List on Main Screen • Show position number of each Sub-Component • Show Component Quantities

To access individual functions shown in the table, use the layer for the relevant component:

Function	Navigation	More Information
Manually add a component to the configuration	Choose *Add Component*	The system displays the list of components that can be added to the configuration manually, with the following information: • Minimum quantity (*How Many Must I Have?*) • Maximum quantity (*How Many Can I Have?*) • Current quantity (*How Many Do I Have?*) This information is maintained in the super BOM. Choose the *Add Component* pushbutton next to the component you want to add. When you have completed your entries, choose *Return to Configuration* to continue.
Delete a component	Choose *Delete Component*	You cannot delete a component that is related to another component by a constraint.
Specialize a component	Choose *Specialize*	The system displays a list of the possible products. The list is maintained as part of the knowledge base for the solution.
Unspecialize a component	Choose *Unspecialize*	You use this function to reverse a specialization.

3.8 Restoring Solution Configuration

The Configuration Engine processes data used during configuration of products. It also supports configuration processes and the final configuration can be saved and restored later, for editing.

The restore process creates/modifies facts, fires dependencies, and then deletes the facts used while configuring the products. For more information about this process, refer to SAP Note 2365244 (Configuration restore based on the user inputs).

3.9 Configuration Result Schema Definition

While performing configuration in SSC, the configuration result information is captured in an XML document, which is then submitted to the integrated backend system for further processing

In some custom integration scenarios, it might be required to use this configuration xml for some custom process flows. Refer to the attachment `SSC_ConfigResult_1.0.xsd` in SAP Note 2948568 for the XML schema definition of the SSC configuration result.

4 User Exit Management

User exits are user-defined functions (declarative functions and pfunctions in the configuration engine and pricing formulas in the pricing engine) which consist of customer-specific code that allows specific custom tasks to be performed. User-defined functions are invoked from a model during a configuration session. They can be used, for example, for checking values and inferring characteristic values.

There are two types of user-defined function, namely declarative functions and procedural functions (pfunctions). From a technical perspective, both are implemented in the same way. They are Java classes that implement a specific interface. During a configuration session, these Java classes must be made available to the configuration runtime engine, which happens through an automated process.

Storage

Depending upon the deployment scenario, user exits are stored in a central database of AS ABAP (ECC/CRM) or AS Java, and can be uploaded without having to interrupt and restart the application.

Management

User exits are managed in a central location. This involves uploading new exits, deleting existing ones, and retrieving information about the exits currently deployed. Management takes place via the REST API interface. This REST API can be called from different locations, for example, from a third-party application. The solution modeling environment uses this REST API interface in its pFunction wizard.

Units and Version

All java classes of a user exit are confined within a unit and can be identified with a *Unit Name* during the upload process. Like in a java project, java classes are expected to be unique in a unit. As solutions evolve, it might be required to test the behavior of a new version of a user exit. This can be achieved by specifying the *Version* field during upload of the user exit, for example, **`dev`** , **`test`** , **`prod`**. The configuration runtime engine can use only one *version* at a time. This is specified in the configuration parameters of the engine settings and can be configured in the server.

The version column is of type *string*, and is limited to 30 characters.

4.1 Deployment Modes

The following two modes of pFunction deployment are supported in SSC:

- **SME pFunction Wizard**
  A new pFunction wizard has been provided in SME to facilitate quick development of user exits. For testing an SSC configuration in the SME test perspective, pFunctions are required to be exported to a local database. For use in NetWeaver Java or SAP Commerce integrated with SAP backend system, pFunctions are required to be exported to the connected backend system. For use in NetWeaver Java or SAP Commerce without any integration with SAP backend systems, pFunctions are required to be exported to the SSC local database, which is same as export to the local database mechanism in SME.
- **Manual Upload in Backend**
  In some situations, it might be required to upload the pFunctions in the backend manually. A new utility has been added in `CFG_SUPPORT` for the same.

4.2 Invocation

User exits are invoked based upon the *version* setting of the configuration runtime engine parameter. One version can be active at any given point in time in SME.

- **Netweaver Java connected to backend CRM, ECC or S/4 Hana systems**
  At the time of configuration invocation, based upon the version setting in engine.xml, SSC fetches all the pFunctions that exist in backend system and caches them locally in the NW Java global cache. It also ensures that any updated pFunctions are considered when a new configuration is launched.
- **Netweaver Java connected to a database in Ready-to-Integrate scenario**
  pFunctions are fetched from the local database and cached locally in NW java in a similar way as the connected backend scenario.
- **SAP Commerce connected to backend ECC or S/4 HANA system**
  pFunctions are replicated to the SSC database in SAP commerce via a data loader during the initial download process via customizing download. In case, when a pFunction is updated after the initial download process has completed, and the data loader is running in delta mode, the updated pFunctions can be replicated again via the *Request Mode* process. Similar to the caching process in NetWeaver java, pFunctions are fetched from the SSC database and cached in SAP commerce for use by different configuration sessions. Apart from standard integrations, it is possible to override and invoke a specific pFunction version by passing its value in the config session context parameter via ejb or REST interface.

4.3 Security Considerations

Role Assignments

Only users having the SME and EJB IPC roles will be able to upload pFunctions in the backend system.

Size Limitations

The upload file size has been defaulted to a maximum of 20MB, and a file size exceeded exception will be thrown when this limit is exceeded. This can be controlled via the customizing settings in backend or by changing the engine settings, in case of export to local database or NW Java for Ready-to-Integrate scenario.

Virus Scanning

For protection against any malicious software, virus scanning can be enabled in NW ABAP or NW Java. For more information, see Enabling Virus Scanner for User Exits in the SSC Security Guide on the SAP Help Portal.

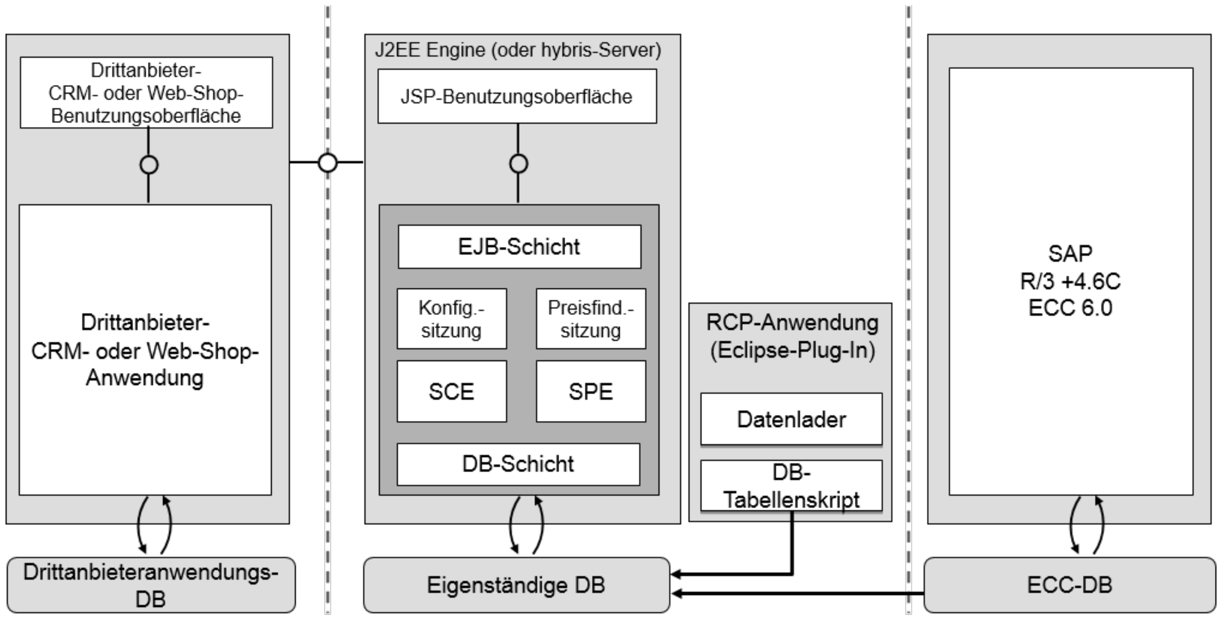
5 Integration with Other Sales Systems

Use

SAP Solution Sales Configuration can be integrated with other sales systems. This scenario is relevant for SAP customers who use their own or a third-party Customer Relationship Management (CRM) system or an e-commerce application, and wish to use the configuration and pricing features of SAP Solution Sales Configuration outside the standard SAP scenarios.

Integration

The following figure illustrates the architecture for integrating SAP Solution Sales Configuration with Hybris and non-SAP sales systems.


Drittanbieter-CRM- oder Web-Shop-Benutzungsoberfläche
Drittanbieter-CRM- oder Web-Shop-Anwendung
Drittanbieteranwendungs-DB
J2EE Engine (oder hybris-Server)
JSP-Benutzungsoberfläche
EJB-Schicht
Konfig.-sitzung
Preisfind.-sitzung
SCE
SPE
DB-Schicht
Eigenständige DB
RCP-Anwendung (Eclipse-Plug-In)
Datenlader
DB-Tabellenskript
SAP R/3 +4.6C ECC 6.0
ECC-DB

Architecture for Integrating SAP Solution Sales Configuration and Non-SAP Sales Systems

SAP Solution Sales Configuration consists of a sales configuration engine (SCE), a sales pricing engine (SPE), and a configuration user interface (UI) based on JavaServer Pages (JSP). A component called the Data Loader is used to fill a standalone database with configuration data from a source ERP Central Component (ECC) system. The system provides application programming interfaces (APIs) that enable third-party CRM and Web Shop applications to interact with the configuration engine, pricing engine, and JSP UI.

Features

- **Configuration APIs**
  APIs are provided for implementing the following configuration functions:
  - Creation of a configuration session.
  - Interaction with the configuration session for compatible mode, advanced mode, and knowledge base orchestration.
  - Retrieval of the configuration result (in BLOB or XML format).
  - Retrieval of the sales structure (order mapper).
- **Pricing APIs**
  APIs are provided for implementing the following pricing functions:
  - Creation of a pricing session.
  - Interaction with a pricing session.
  - Retrieval of the pricing result (in BLOB or XML format).
- **Deployment as a J2EE Application**
  The configuration engine is built and packaged as an Enterprise JavaBeans (EJB) component based on Java 2 Enterprise Edition (J2EE) standards. It is primarily intended to be deployed on an SAP J2EE engine (SAP NetWeaver 7.30) but can be adapted to run on any J2EE-certified runtime environment (Java 6) with minimum implementation effort.
- **Deployment as a Desktop Application**
  The configuration engine can be deployed as libraries that can be embedded in a desktop application. The EJB layer can be by-passed to allow direct access to the configuration session or to the engine abstraction layer of the application.
- **Enhanced JSP UI**
  The JSP UI can be called with an HTTP request.
- **Standalone or Local Database**
  The configuration engine can be connected directly to a database management system such as Microsoft SQL Server. The system provides scripts to create the database and an application to run the scripts.
- **Data Loader**
  The Data Loader is used to fill the local standalone database with configuration data from an ECC system. For more information, see Data Loader in the Solution Modeling Environment [page 109].

Constraints

In the solution modeling environment, the Data Loader does not download pricing data or customizing data. The interactive pricing and delta pricing features do not, therefore, work if the configuration environment is used with a standalone database.

If the configuration environment is connected to the ECC database, the interactive pricing and delta pricing features work, and the standalone database and the Data Loader are not required.

6 Business Functions

This section provides information about the various business functions that are available in SAP Solution Sales Configuration.

6.1 Compressed Storage of XML Configuration Results

Use

Technical Data

Technical Name of Business Function	`/SLCC/BF_XML_COMPRSSION` `/SLCE/BF_XML_COMPRSSION`
Type of Business Function	Enterprise Business Function
Available From	SAP Solution Sales Configuration Support Package 3
Application Component	*FBS Solution Configuration* (CRM-SLC) *FBS Solution Configuration* (LO-SLC)
Required Business Function	Not relevant

You can use this business function to store XML configuration results in the database in compressed form. If the business function is deactivated, the configuration results are stored in uncompressed form.

Integration

The business function can be activated separately in SAP ERP and SAP CRM. You can activate the business function in one system but leave it deactivated in the other system.

Prerequisites

- You have installed the following components as of the version mentioned:

Type of Component	Component	Required for the Following Features Only
Software Component	`SLCC` (for SAP CRM) `SLCE` (for SAP ERP)	

- If sales documents with configurable products already exist before you activate the business function, you must migrate the data manually from table `/SLCC/CFG_XML` to `/SLCC/XML_BLOB` (for SAP CRM) or from `/SLCE/CFG_XML` to `/SLCE/XML_BLOB` (for SAP ERP). Unless you do so, the system cannot restore the configurations for existing sales document items.
  The data migration is necessary because XML configuration results for sales document items are no longer stored in table `/SLCC/CFG_XML` or `/SLCE/CFG_XML` after you have activated the business function. Instead, they are stored in tables `/SLCC/XML_BLOB` and `/SLCE/XML_BLOB`. Configuration results are read either from table `*CFG_XML` or `*XML_BLOB` (for example, when the configuration screen of an existing item is opened).

Features

Compression of XML Configuration Results

Configuration results for sales document items are stored as XML in the back-end system. In releases of SAP Solution Sales Configuration earlier than Support Package 3, the configuration results are stored in an uncompressed form. This is also the case in Support Package 3 if this business function is deactivated. If you activate this business function, the configuration results are stored in a compressed form.

If you do not activate the business function, configuration results are stored in table `/SLCC/CFG_XML` for SAP CRM and `/SLCE/CFG_XML` for SAP ERP. If you activate the business function, the configuration results are stored in table `/SLCC/XML_BLOB` for SAP CRM and `/SLCE/XML_BLOB` for SAP ERP.

The business function is reversible and can be deactivated again if it has been activated. Note that some manual migration activities are required for existing configuration results.

7 Solution Configuration Analytics

Use

SAP Solution Sales Configuration allows you to create a variety of complex solution configurations in a hierarchical manner. This flexibility makes the implementation of a standard analytical reporting solution challenging.

For analytical reporting, the system must assemble a large number of objects such as sales order items or quotation items. Furthermore, these must be assembled in accordance with a common set of properties (characteristics) and a common set of measures (key figures). However, if the actual configurations of the solution differ significantly between different sales documents, it becomes difficult for the system to identify the common properties and measures. Therefore, the analytical reporting solution for SAP Solution Sales Configuration is designed to be flexible, so that you can adapt it to suit your own specific analytical requirements.

To analyze the solution selling process, solution configuration data from the sales system must be combined with data from other systems, such as data from the fulfillment process and financial data. Therefore, SAP Solution Sales Configuration requires the use of a separate data warehouse to consolidate the data from the various processes. Data extractors are provided to transfer the data from the source systems to the data warehouse. You can then design your own persistence layer and user interfaces based on your own specific reporting requirements.

Integration

The analytical reporting solution is integrated with the following components:

- SAP Customer Relationship Management (SAP CRM)
- SAP Business Information Warehouse (SAP BW)

> i Note
>
> You can use an alternative data warehouse solution to SAP BW. In this case, you must implement BusinessObjects Data Services to use the data extraction layers.

Features

The system provides a comprehensive set of data extractors in the SAP CRM system. The data extractors allow you to extract all of the relevant solution configuration data to the SAP BW system. The data includes business document item information for the solution configuration, in addition to master data and description data. You can choose the data you consider relevant for analytical reporting.

The DataSources have the following features:

- **Knowledge Base Dependent and Knowledge Base Independent Analytics**
  Periodically, you may need to change your solution models; new features can be added to solutions and other features can be removed. Therefore, the system allows you to create new versions of a solution model, so that after a period of time, each solution can have several different model versions. These different models are known as knowledge bases, and each knowledge base can have several knowledge base run time versions.
  This means that transactional data such as quotation items or sales order items can refer to different knowledge bases even though they refer to the same solution. However, from a business perspective, items for the same solution must not be treated separately if they refer to different knowledge bases. Therefore, the system is capable of recognizing common properties in different knowledge bases of the same solution. If a characteristic exists in a knowledge base of a solution, and if the same characteristic exists in other knowledge base versions, then for analytical purposes, it is possible to provide one common characteristic for all of the different knowledge base versions. This is referred to as **knowledge base independent** analytics.
  The system also allows you to analyze the data in a specific version of the knowledge base. This is referred to as **knowledge base dependent** analytics.
- **Delta Data Extraction**
  The DataSources use a time stamp based delta extraction process to upload only data that has changed since the last upload process. The system also controls the size of the DataSources' internal packages to ensure that the data extraction process is controlled, and that performance is as efficient as possible.

For more information, see DataSources [page 197].

i Note

SAP Solution Sales Configuration does not provide any SAP BW business content objects. You must create these objects based on your own specific analytical requirements.

Constraints

- SAP CRM provides standard DataSources for most CRM Business Documents such as Quotation Items and Sales Order Items. Business Intelligence (BI) content for SAP CRM is also available; this provides InfoProviders and the corresponding staging processes in SAP BW for most SAP CRM objects. The DataSources for SAP Solution Sales Configuration only refer to properties that are specific to solution configuration; properties that are already available in the standard SAP CRM DataSources are not included.
- It is assumed that the meaning of characteristics and symbol values remains unchanged when they are inherited from the previous version of the knowledge base and used in a new solution model. This is required to ensure consistency in knowledge base independent analytics.
- It is recommended that properties are not deleted; instead, they are marked as obsolete. This is because a deleted property can be referenced in historical transactional data. Unless configuration master data has previously been extracted to SAP BW, the analytics layer cannot interpret such properties, and such data is flagged as corrupted in the data extraction layer.

7.1 SAP Business Information Warehouse Implementation

Use

You use this process to create an SAP BW system for the analysis of solution configuration data.

Process

1. You create an extraction layer (data sources) in the SAP Customer Relationship Management (SAP CRM) system.
   SAP Solution Sales Configuration provides a set of data sources for transactional data and master data. These data sources are used to transfer solution configuration data from the SAP CRM system to the SAP BW system. For more information, see DataSources [page 197].
2. You create a persistence layer (data staging) in the SAP BW system.
   In the SAP BW system, you define InfoObjects (for master data) and Data Store Objects or InfoCubes (for transactional data) to store the solution configuration data. You then define Transformations and Data Transfer Processes to transfer the SAP CRM data into the SAP BW persistence layer.

   > ⚠ Caution
   >
   > When defining the persistence layer, you must ensure to take account of delta processing. In particular, the deletion of objects such as order items or characteristics in the SAP CRM system must be properly reflected in the SAP BW system.

3. You create the user interfaces (reports and dashboards).
   You can create a variety of user interfaces such as Queries, Web Templates, Dashboards, Information Spaces in BusinessObjects Explorer, and so on. Most of these interfaces use BW Queries. However, a few BusinessObjects applications such as BusinessObjects Explorer or Web Intelligence use a different method to access SAP BW data. You can create BW Queries on the Data Store Objects or the InfoCubes that have been used for the staging process, or you can define additional InfoProviders, depending on your reporting requirements. You can also create InfoProviders without persistency in SAP BW (MultiProviders and InfoSets), if necessary.

> i Note
>
> SAP Solution Sales Configuration does not provide business intelligence (BI) content for the persistence layer and the user interface layer. You must create these based on your own reporting requirements.

7.2 DataSources

The data model for solution configuration data has the following elements:

- **Characteristics**

The properties of a solution are called characteristics.

- **Knowledge Bases**
  All of the characteristics of a solution are stored in a knowledge base. Each solution can have several knowledge bases versions. In the SAP Customer Relationship Management (SAP CRM) user interface, these knowledge bases are called *Product Models*. Each knowledge base is uniquely identified by the Knowledge Base Name, the Knowledge Base Version, and the Logical System. In the SAP CRM system, a knowledge base is identified by an integer value called a knowledge base ID (KBID) in table CRMM_CFGKB.
- **Symbol Values**
  Each characteristic has a set of possible values known as symbol values.

The system provides one text DataSource and one attribute DataSource for each of the model elements (knowledge bases, characteristics, and symbol values). There are two versions of each of these DataSources: one for knowledge base dependent analytics and one for knowledge base independent analytics.

7.2.1 Item Configuration for Business Documents

DataSource Transactional Data `/SLCC/CFG_ORDER_ITEM`

Use

You use this DataSource to extract configuration information on a sub item level from SAP Customer Relationship Management (SAP CRM) business documents. The selection parameters are OBJECT_TYPE and PROCESS_TYPE. For convenience, OBJECT_ID is also included as a selection parameter, so that the result of an extraction can be tested in transaction RSA3 (Extractor Checker) for individual SAP CRM Business Documents.

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	Yes

Data Modeling

Delta Update

The delta management is based on the time stamp field HEAD_CHANGED_AT in CRMD_ORDERADM_H. Since items with configurations have an entry in table CRMD_STRUCT_I, all such items can be retrieved with an inner join between CRMD_ORDERADM_H and CRMD_STRUCT_I. To efficiently control the package size the following strategy is applied:

1. All items with configuration data from CRMD_ORDERADM_H and CRMD_STRUCT_I are stored in a static table in the first package. The table is sorted by the header GUID.
2. From the internal item table, a minimum number of items are processed (currently 100). To ensure that all the items in a business document are processed in one package, items are added from the internal item table until a new header GUID is encountered.
3. For all these items the configuration data is retrieved from the cBase using the function module COM_CUXI_GET_MULTI_CFG and then the extract table is populated. The current size of the package is computed and if it is less than the minimum package size, then the process is repeated.

Subsequently, the additional fields in the extract structure for knowledge base independent analytics are populated using the function modules /SLCC/READ_KBID_I and /SLCC/READ_CHARAC_I.

Configuration instances can have references to objects that have been deleted from the knowledge base. In this event, the indicator field CORRUPTED in the extract structure is set to X (otherwise the value is a space).

⚠ Caution

This DataSource reads directly from the database tables CRMD_ORDERADM_H and CRMD_STRUCT_I. These tables provide information about the currently existing Business Document data. They do not contain any information about deleted items or deleted documents. If previously uploaded data is deleted, then such data is not automatically deleted in the SAP Business Information Warehouse (SAP BW). This can only be achieved in the SAP BW staging process using the standard SAP Business Intelligence (SAP BI) content DataSources such as 0CRM_SALES_ORDER_I. In delta uploads, only the One Order DataSources contain the necessary information about deleted objects. The design of a proper staging process is described in the How-To Guide for SAP Solution Sales Configuration analytics.

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`GUID`	GUID of an SAP CRM Order Object		
`HEADER`	GUID of an SAP CRM Order Object		
`HEAD_CHANGED_AT`	Time of last change to the transaction		
`OBJECT_ID`	Transaction ID		
`OBJECT_TYPE`	Business transaction cate-gory		
`PROCESS_TYPE`	Business transaction type		

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
PARENT	GUID of an SAP CRM Order Object		
PRODUCT	Internal unique ID of a product		
ORDERED_PROD	Product name entered		
DATE_KB_DETERM	Date for knowledge base determination in IPC (master data version configuration)		
KBNAME	Knowledge base object		
LOGSYS	Logical system		
KBVERSION	Runtime version of solution configuration engine knowledge base		
KBID	Internal identifier of a knowledge base		
KBID_I	Internal identifier of a knowledge base		
OBJ_TYPE	CUIB: External type of referencing object		
CHARC	Characteristic name		
CHARID	Internal identifier of a model element		
CHARID_I	Internal identifier of a model element		
DATATYPE	Characteristic data type (for example, string or float)		
VACHARC	Characteristic name		
VARCHARID	Internal identifier of a model element		
VCHARID_I	Internal identifier of a model element		
VALUE	Characteristic value		
AUTHOR	Statement was inferred		
VDATATYPE	Characteristic data type (for example, string or float)		
FACTOR	Factor		
QUANTITY	Solution Configurator: Quantity		
UNIT	Solution Configurator: Quantity Unit		

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
CORRUPTED	Single-character flag		
RECORDMODE	BW Delta Process: Record Mode		
LINK_GUID	GUID of an SAP CRM Order object		

7.2.2 Knowledge Base Attributes

/SLCC/CFG_KBID_ATTR

Use

You use this DataSource to extract data from COMM_CFGKB, based on a delta time stamp in /SLCC/CFGKB_CHGD.

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
KBID	Internal identifier of a knowledge base		
KBID_I	Internal identifier of a knowledge base		
LOGSYS	Logical system		
KBOBJNAME	Knowledge base name		
VERSION	Knowledge base version		
FROMDATE	Knowledge base validity date		
TODATE	Knowledge base validity date		
CREATED_BY	Knowledge base creator		
CREATED_AT_DAY	Knowledge base validity date		
CHANGED_BY	Knowledge base changer		
CHANGED_AT_DAY	Knowledge base validity date		
CHANGED_AT	Time of last change to the transaction		

7.2.3 Knowledge Base Descriptions

/SLCC/CFG_KBID_TEXT

Use

You use this DataSource to extract data from COMM_CFGKBTX, based on a delta time stamp in /SLCC/CFGKB_CHGD.

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	

Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`KBID`	Internal identifier of a knowledge base		
`LANGU`	Language key		
`TXTLG`	Long description		
`CHANGED_AT`	Time of the last change to the transaction		

7.2.4 Knowledge Base Characteristic Attributes

`/SLCC/CFG_CHARAC_ATTR`

Use

You use this DataSource to extract knowledge base characteristic attributes from COMM_CFGCHARAC and COMM_CFGCHAREF, based on a delta time stamp in /SLCC/CFGKB_CHGD. The extraction takes place by way of the structure /SLCC/S_CFG_CHARAC_ATTR, using the extractor /SLCC/CFG_CHARAC_ATTR.

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	

Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
KBID	Internal identifier of a know-ledge base		
CHARID	Internal identifier of a model element		
CHARNAME	Characteristic name (object characteristic)		
DATATYPE	Characteristic data type (for example, string or float)		
TABLENAME	Characteristic table name (object characteristic)		
FIELDNAME	Characteristic field name (object characteristic)		
CHANGED_AT	Time of last change to the transaction		

7.2.5 Knowledge Base Characteristic Descriptions

`/SLCC/CFG_CHARAC_TEXT`

Use

You use this DataSource to extract description data from COMM_CFGCHATX, based on a delta time stamp in /SLCC/CFGKB_CHGD.

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`KBID`	Internal identifier of a know-ledge base		
`CHARID`	Internal identifier of a model element		
`LANGU`	Language key		
`TXTLG`	Long description		
`CHANGED_AT`	Time of last change to the transaction		

7.2.6 Knowledge Base Symbol Value Attributes

`/SLCC/CFG_SYMVAL_ATTR`

Use

You use this DataSource to extract data from COMM_CFGSYMVAL. The usage is then determined from the following tables:

- COMM_CFGSTRDOM
- COMM_CFGNUMDOM

- COMM_CFGCLSDOM
- COMM_CFGCLNDOM

Numerical values are used to define ranges in the application, and indicators are used to define whether each numerical value is the start, or the end of the range, or both. VALUECODE is populated with the usage type. For example, the value type `01` means that the symbol value is used in the application with meaning `equal`.

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`KBID`	Internal identifier of a know-ledge base		
`CHARID`	Internal identifier of a model element		
`KBID_I`	Internal identifier of a know-ledge base		
`CHARID_I`	Internal identifier of a model element		
`DATATYPE`	Characteristic data type (for example, string or float)		
`VALUE`	Characteristic value		
`VALUECODE`	Value code		
`TO_INDICATOR`	Boolean variable (X=True, -=False, Space=Unknown)		

7.2.7 Knowledge Base Symbol Value Descriptions

/SLCC/CFG_SYMVAL_TEXT

Use

You use this DataSource to extract description data from COMM_CFGVALTX for string-like symbols, based on a delta time stamp in /SLCC/CFGKB_CHGD.

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`KBID`	Internal identifier of a know-ledge base		
`CHARID`	Internal identifier of a model element		
`VALUE`	Characteristic value		
`LANGU`	Language key		
`TXTLG`	Long description		
`CHANGED_AT`	Time of last change to the transaction		

7.2.8 Knowledge Base Independent Attributes

`/SLCC/CFG_KBID_I_ATTR`

Use

You use this DataSource to extract knowledge base independent attributes from COMM_CFGKB. The properties are retrieved from the knowledge base with the most recent validity date (the current version of the knowledge base).

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`KBID_I`	Internal identifier of a know-ledge base		
`LOGSYS`	Logical system		
`KBOBJNAME`	Knowledge base name		
`CREATED_AT_DAY`	Knowledge base validity date		
`CHANGED_AT_DAY`	Knowledge base validity date		
`CHANGED_AT`	Time of last change to the transaction		

7.2.9 Knowledge Base Independent Descriptions

`/SLCC/CFG_KBID_I_TEXT`

Use

You use this DataSource to extract knowledge base independent descriptions from COMM_CFGKBTX. The properties are retrieved from the knowledge base with the most recent validity date (the current version of the knowledge base).

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`KBID_I`	Internal identifier of a know-ledge base		
`LANGU`	Language key		
`TXTLG`	Long description		
`CHANGED_AT`	Time of last change to the transaction		

7.2.10 Knowledge Base Independent Characteristic Attributes

`/SLCC/CFG_CHARAC_I_ATTR`

Use

You use this DataSource to extract knowledge base independent characteristic attributes. The properties are retrieved from the knowledge base with the most recent validity date (the current version of the knowledge base). The extraction takes place by way of the structure /SLCC/S_CFG_CHARAC_I_ATTR, using the extractor /SLCC/CFG_CHARAC_I_ATTR.

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`KBID_I`	Internal identifier of a know-ledge base		
`CHARID_I`	Internal identifier of a model element		
`CHARNAME`	Characteristic name (object characteristic)		
`DATATYPE`	Characteristic data type (for example, string or float)		

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`TABLENAME`	Characteristic table name (object characteristic)		
`FIELDNAME`	Characteristic field name (object characteristic)		
`CHANGED_AT`	Time of last change to the transaction		

7.2.11 Knowledge Base Independent Characteristic Descriptions

`/SLCC/CFG_CHARAC_I_TEXT`

Use

You use this DataSource to extract knowledge base independent characteristic descriptions. The properties are retrieved from the knowledge base with the most recent validity date (the current version of the knowledge base). The extraction takes place by way of the structure `/SLCC/S_CFG_CHARAC_I_TEXT` using the extractor `/SLCC/CFG_CHARAC_I_TEXT`.

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
KBID_I	Internal identifier of a knowledge base		
CHARID_I	Internal identifier of a model element		
LANGU	Language key		
TXTLG	Long description		
CHANGED_AT	Time of last change to the transaction		

7.2.12 Knowledge Base Independent Symbol Value Attributes

/SLCC/CFG_SYMVAL_I_ATTR

Use

You use this DataSource to extract knowledge base independent symbol value attributes from COMM_CFGSYMVAL. The properties are retrieved from the knowledge base with the most recent validity date (the current version of the knowledge base).

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
KBID_I	Internal identifier of a know-ledge base		
CHARID_I	Internal identifier of a model element		
DATATYPE	Characteristic data type (for example, string or float)		
VALUE	Characteristic value		
VALUECODE	Value code		
TO_INDICATOR	Boolean variable (X=True, -=False, Space=Unknown)		

7.2.13 Knowledge Base Independent Symbol Value Descriptions

/SLCC/CFG_SYMVAL_I_TEXT

Use

You use this DataSource to extract knowledge base independent symbol value descriptions from COMM_CFGVALTX. The properties are retrieved from the knowledge base with the most recent validity date (the current version of the knowledge base).

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	Yes

Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`KBID_I`	Internal identifier of a know-ledge base		
`CHARID_I`	Internal identifier of a model element		
`VALUE`	Characteristic value		
`LANGU`	Language key		
`TXTLG`	Long description		
`CHANGED_AT`	Time of last change to the transaction		

7.2.14 Configuration Value Code Descriptions

`/SLCC/CFG_VALUECODE_TEXT`

Use

You use this DataSource to extract configuration value code descriptions from the /SLCC/CFG_VALUECODE domain. The extraction takes place by way of the structure *Interface: Generic transfer of text* (ROTEXTSTR1).

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist

RemoteCube-Capable	No
Delta-Capable	No
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`LANGU`	Language		
`KEY1`	Key field		
`TXTMD`	Medium description		

7.2.15 Configuration Data Type Descriptions

`/SLCC/CFG_DATATYPE_TEXT`

Use

You use this DataSource to extract configuration data type descriptions from the /SLCC/CFG_DATATYPE domain. The extraction takes place by way of the structure *Interface: Generic transfer of text* (ROTEXTSTR1).

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	No
Extraction from Archives	No

Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`LANGU`	Language		
`KEY1`	Key field		
`TXTSH`	Short description		

7.2.16 Configuration Author Descriptions

`/SLCC/CFG_AUTHOR_TEXT`

Use

You use this DataSource to extract configuration author descriptions from the /SLCC/CFG_AUTHOR domain. The extraction takes place by way of the structure *Interface: Generic transfer of text* (ROTEXTSTR1).

Technical Data

Application Component	SAP Solution Sales Configuration (CRM-SLC)
Exchange Available as of Release	1.0
Shipment	
Content Versions	No Content versions exist
RemoteCube-Capable	No
Delta-Capable	No
Extraction from Archives	No
Verifiable	No

Data Modeling

Fields of Origin for the Extraction Structure

Fields in the Extraction Structure	Description of the Field in the Extraction Structure	Table of Origin	Field in the Table of Origin
`LANGU`	Language		
`KEY1`	Key field		
`TXTMD`	Medium description		

8 Operations Information

Certain administrative activities are required to use SAP Solution Sales Configuration. For more information about the use and administration of the application, see the following sections:

Related Information

Monitoring Concept [page 218]
Logging and Tracing [page 218]
Cache Management [page 220]
Frequently Asked Questions [page 223]

8.1 Monitoring Concept

Use

Wily Introscope is a third-party administrative tool from Computer Associates. It is used for application performance monitoring and diagnostics, and can be integrated with SAP Solution Sales Configuration.

More Information

For more information about Wily Introscope, see http://wiki.sdn.sap.com/wiki/display/TechOps/RCA_Home..

For information about the installation, setup, and configuration of Wily Introscope, see SAP Notes 797147 (Introscope Installation for SAP Customers) and 1237887 (Introscope 7.2.3 Release Notes).

8.2 Logging and Tracing

You can implement logging and tracing in SAP Solution Sales Configuration using the standard logging capabilities provided in the Java Development Kit (JDK). The package `java.util.logging` provides logging capabilities via the class Logger.

After installing the solution modeling environment plug-in in Eclipse, you can enable logging in the `eclipse.ini` file.

Alternatively, you can use the general Java util logging settings. For more information, refer to the Java logging application programming interface (API) documentation.

SLG1 Logging

You can check SSC log details in transaction `SLG1`, for the following components:

- `/SLCC/ (CRM)`
- `/SLCE/ (ECC)`

Java Netweaver Logging

The application (session) log is important for the understanding and initial analysis of the issue as well as the technical flow of the process.

You can create an application log using the instructions in SAP Note 1090753 (Creation of logs for WebChannel/E-Commerce applications).

You can create a session-specific log using the instructions in SAP Note 921409 (Enable session tracing in mySAP CRM 5.0 java components).

8.3 Setting the User Exit Upload Size Limit in CRM/ECC

You can configure the maximum size limit for the uploaded user exit (.jar) files. To do this, perform the following steps:

1. In the ABAP backend system, a new transaction code `/SLCE/FILE_SIZE` has been delivered by SSC. Execute this transaction code to maintain the maximum permissible size limit for the uploaded file.
2. When you enter the transaction, you need to maintain a single integer entry in the view. This value will correspond to the size limit of the uploaded file in MBs (megabytes).
3. Save your entries.

How this works?

Considering an entry has been maintained in the above view, when we upload a file using the Solution Modeling Environment (SME), the ABAP back end enforces this size limit and if the size of the file exceeds this value, `INSERT_EXCEPTION`, a kind of exception is returned to the SME, and a message is logged in the ABAP SLG1 application logs as follows:

Object: `/SLCE/` or `/SLCC/`

Sub-object: `DATALODER`

Message: `User exit upload failed. Size for the filename & exceeds the limit & MB.`

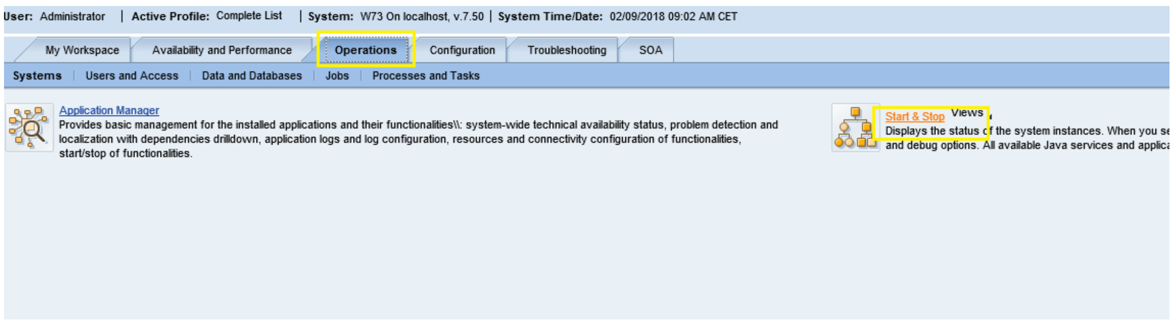
DEFAULT VALUES: If the end user has not maintained any value in this field, a default value of 20 MB is taken as the maximum permissible file size.

8.4 Cache Management

Clearing SAP Solution Sales ConfigurationRuntime Engine Cache on NetWeaver Java Server

You can clear the SAP Solution Sales Configurationruntime engine cache on the NetWeaver Java server using the procedure below:

1. Log in to *NW* > *Operations* > *Start & Stop*



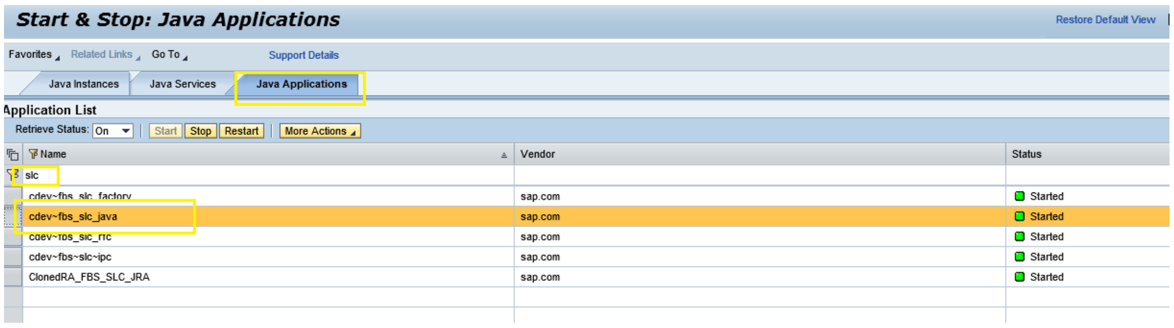
User: Administrator | Active Profile: Complete List | System: W73 On localhost, v.7.50 | System Time/Date: 02/09/2018 09:02 AM CET

Application Manager
Provides basic management for the installed applications and their functionalities\\: system-wide technical availability status, problem detection and localization with dependencies drilldown, application logs and log configuration, resources and connectivity configuration of functionalities, start/stop of functionalities.

Start & Stop Views
Displays the status of the system instances. When you se
and debug options. All available Java services and applica

Logging on to NW

2. Once you click on *Start & Stop*, go to the *Java Applications* tab
3. Search for keyword **slc** and select the component *cdev~fbs_slc_java*



Start & Stop: Java Applications

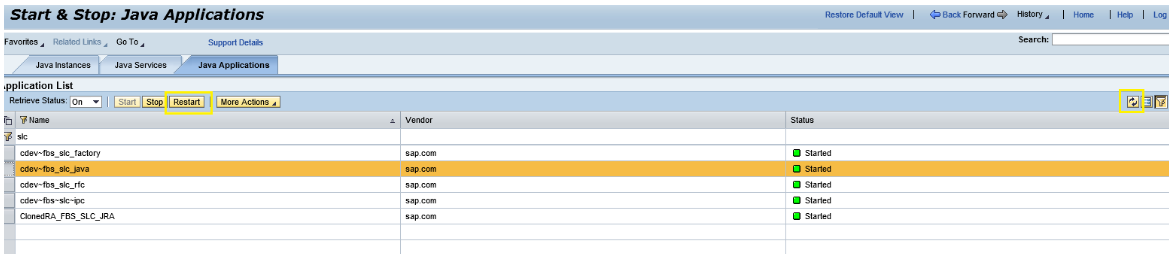
Application List

Retrieve Status: On

Name	Vendor	Status
slc		
cdev~fbs_slc_factory	sap.com	🟩 Started
cdev~fbs_slc_java	sap.com	🟩 Started
cdev~fbs_slc_rfc	sap.com	🟩 Started
cdev~fbs~slc~ipc	sap.com	🟩 Started
ClonedRA_FBS_SLC_JRA	sap.com	🟩 Started

Selecting the Component

4. Click on *Restart* to refresh to clear the cache


Start & Stop: Java Applications
Restore Default View | Back Forward | History | Home | Help | Log
Favorites Related Links Go To Support Details
Search:
Java Instances
Java Services
Java Applications
Application List
Retrieve Status: On
Start
Stop
Restart
More Actions

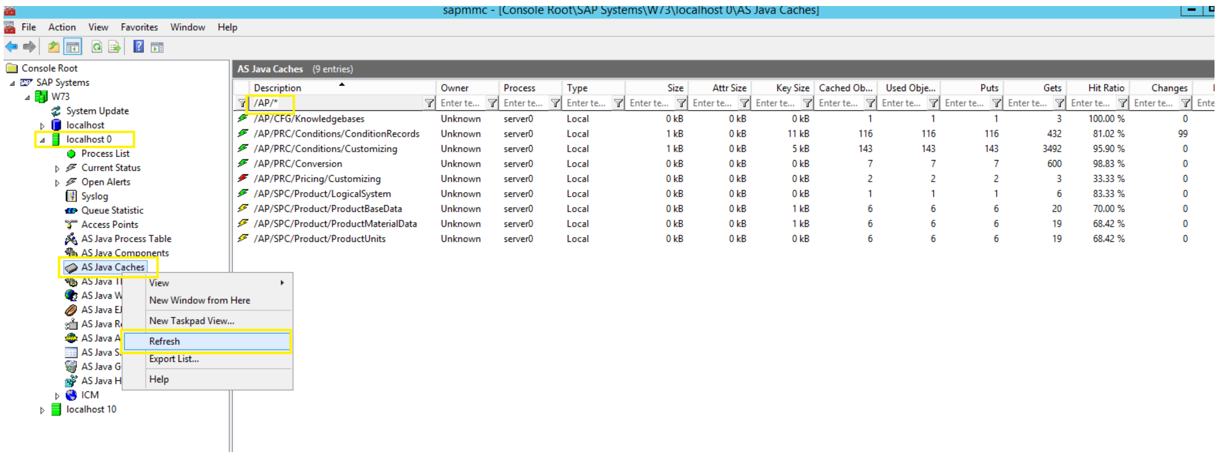
Name	Vendor	Status
slc		
cdev~fbs_slc_factory	sap.com	🟩 Started
cdev~fbs_slc_java	sap.com	🟩 Started
cdev~fbs_slc_rfc	sap.com	🟩 Started
cdev~fbs~slc~ipc	sap.com	🟩 Started
ClonedRA_FBS_SLC_JRA	sap.com	🟩 Started

Refreshing the Cache

Monitoring SAP Solution Sales ConfigurationRuntime Engine Cache on SAP Management Console

You can monitor SAP Solution Sales Configuration runtime engine cache on SAP management console using the procedure below:

1. Go to *Console Root* > *SAP Systems* > *W73* > *localhost0* > *AS Java Caches*
   Right click and *Refresh*.
2. In the details window, filter the list of *AS Java Caches* by **/AP/*** to view further details of the relevant caches



Description	Owner	Process	Type	Size	Attr Size	Key Size	Cached Ob...	Used Obje...	Puts	Gets	Hit Ratio	Changes
/AP/*	Enter te...	Enter te...	Enter te...	Enter te...	Enter te...	Enter te...	Enter te...	Enter te...	Enter te...	Enter te...	Enter te...	Enter te...
/AP/CFG/Knowledgebases	Unknown	server0	Local	0 kB	0 kB	0 kB	1	1	1	3	100.00 %	0
/AP/PRC/Conditions/ConditionRecords	Unknown	server0	Local	1 kB	0 kB	11 kB	116	116	116	432	81.02 %	99
/AP/PRC/Conditions/Customizing	Unknown	server0	Local	1 kB	0 kB	5 kB	143	143	143	3492	95.90 %	0
/AP/PRC/Conversion	Unknown	server0	Local	0 kB	0 kB	0 kB	7	7	7	600	98.83 %	0
/AP/PRC/Pricing/Customizing	Unknown	server0	Local	0 kB	0 kB	0 kB	2	2	2	3	33.33 %	0
/AP/SPC/Product/LogicalSystem	Unknown	server0	Local	0 kB	0 kB	0 kB	1	1	1	6	83.33 %	0
/AP/SPC/Product/ProductBaseData	Unknown	server0	Local	0 kB	0 kB	1 kB	6	6	6	20	70.00 %	0
/AP/SPC/Product/ProductMaterialData	Unknown	server0	Local	0 kB	0 kB	1 kB	6	6	6	19	68.42 %	0
/AP/SPC/Product/ProductUnits	Unknown	server0	Local	0 kB	0 kB	0 kB	6	6	6	19	68.42 %	0

Monitoring SAP Solution Sales ConfigurationRuntime Engine Cache on SAP Management Console

8.5 Solution Configuration Support - Backend Utilities

Use the solution configuration support to access (transaction `/n/SLCE/CFG_SUPPORT` in ECC and `/n/SLCC/CFG_SUPPORT` in CRM) SSC related utilities.

SSC Download XML

Here, you can download the sales data in an XML, simply by entering the relevant document and item numbers. IPC does not store data in an XML but SSC does.

Knowledge Base Preload

Here, you can preload some standard data and inputs for the configuration page, to enhance performance of the tool. The knowledge bases are synced with a cache that is called when the pre-defined set of inputs are entered to generate the configuration page.

You can do this simply by entering the product name, type, its validity date, and the name of the production system.

Initialize Configuration DB

This utility is used to initialize the replication of classical knowledge bases generated in SAP ERP, from the `SCE*` tables to the `COMM_CFG*` tables.

A similar standard utility report `CFG_ERP_INITIALISE_DB` also exists for this purpose, but it is advisable to not use this standard report in SAP Solution Sales Configuration. This standard report deletes all the advanced mode knowledge-base runtime versions from the `COMM_CFG*` tables.

Update Cache for External Text and Variant Tables

Here, you can update the data that is stored in the cache, post updation of the data in the external text and variant tables.

You can identify the specific table for updation and use this utility to process the change in data.

Delete Orphan Configuration XML

Here, you can clean up your system and remove the configuration XMLs that are now obsolete.

You can simply simulate the procedure.

Compare KB Runtime Version

This utility calculates size information for a given set of knowledge base runtime versions.

SME Models STORAGE Migration

For models created with version lower than SME 4.0, this utility is required to update the `KnowledgeBase` table for storage indicator. Models created with SME should have storage = 30 in table `COMM_CFGKB`.

Data Protection and Privacy

This utility provides the following functions:

- Displays all knowledge bases created/changed by a user
- Replace the user name from knowledge base tables
- Displays all pFunction user exits created/changed by a user
- Replace the user name from pFunction user exit tables

Maintain pFunction User Exits

This utility provides the following functions:

- Upload of pFunction user exits in the ERP database by uploading user exits in the form .JAR files.
- Deletion of pFunction user exits already existing in the database table `/SLCE/USER_EXIT`.
- To retrieve an archived user exit version:
  It can happen that a new version of the pFunction user exit can cause some malfunction in the SSC Engine. This feature can be used to retrieve the previously working user exit version (as a .JAR file) to restore normalcy. The archived records are stored in the database table */SLCE/USER_EXITH*.

Find Missing Materials of Knowledge Base

This utility provides a list of missing materials of a knowledge base. If any material is missing but is instantiated in a configuration, errors are raised in the backend. This utility helps to list down all the missing materials in the system that are part of knowledge base definition. Customers can then create these materials to avoid any issues later on.

8.6 Frequently Asked Questions

For more information to use SAP Solution Sales Configuration, refer to the following SAP Notes:

- 2314556 (SAP Solution Sales Configuration - Frequently Asked Questions)
- 1876540 (SAP Solution Sales Configuration (SSC) - Solution Modeling Environment (SME) FAQ)
- 2246019 (SAP Solution Sales Configuration Commerce - Frequently Asked Questions)

Important Disclaimers and Legal Information

www.sap.com/contactsap

THE BEST RUN SAP