Creating Context-Based Adaptations with FPM: Step-By-Step Guide

The option to adapt applications without the need of modification is one of the main benefits offered by Floorplan Manager (FPM).

The possibilities of Web Dynpro Customizing in combination with the Enhancement Framework already allow far more than the classic Dynpro in this respect.

Christian Guenther
## TABLE OF CONTENTS

INTRODUCTION .................................................................................................................. 3
Use-Case Example ................................................................................................................ 3

BASIC CONCEPTS ............................................................................................................ 4
Adaptation Schema .............................................................................................................. 4
Adaptation Dimension ......................................................................................................... 4
Adaptation Context .............................................................................................................. 4
Inheritance of Component Configurations ............................................................................ 4

STEP-BY-STEP EXAMPLE .................................................................................................. 4
Adding an Attachment UIBB for Managers .......................................................................... 4
1. Create the Adaptation Schema ....................................................................................... 4
2. Create the WD Application ............................................................................................ 5
3. Create the Application Configuration ............................................................................. 7
4. Create the CBA Configuration ....................................................................................... 8
5. Create an Adaptation .................................................................................................... 8

Adapting the Address Layout ............................................................................................. 9
1. Extending the Form’s Feeder Class ............................................................................... 9
2. Replacing the Standard Feeder Class ........................................................................... 11
3. Creating an Adaptation for French Addresses ............................................................... 11
4. Enhancing the French Address Adaptation .................................................................. 12

Avoiding Unnecessary FPM Events .................................................................................. 13
Setting the Adaptation Context Locally ............................................................................... 13
Hiding of UIBBs .................................................................................................................. 15
INTRODUCTION

The option to adapt applications without the need of modification is one of the main benefits offered by FPM. The possibilities of Web Dynpro Customizing in combination with the Enhancement Framework already allow far more than the classic Dynpro in this respect.

Nevertheless, these two options still have quite strong limitations: There is only one customizing per client and Enhancements are system-wide active. This allows adapting applications to the need of a customer, but not to the need of different contexts within one customer system.

Use-Case Example

It is quite simple to adapt the screen below by customizing or enhancing. The customer can rearrange or hide fields; he can also add additional fields. However, he must also decide how this screen should look like for all situations. He cannot adapt it depending on some runtime parameter.

There could be the requirement to add additional information to (or remove it from) the screen for particular users (in our example, it has been decided to remove the details assignment block and some other fields to avoid unnecessary load on managers).

Previously, there was only one option to achieve this: copy the whole set of configurations, adapt it and have the particular users start the adapted configurations.

There was also another problem, that is, the need for data-dependent adaptations. In the previous screen, the Street/House Number fields are placed in the right order for German addresses. In France, house number and street name are the other way around. Therefore, an adaptation for French addresses would simply swap those two fields around. However, previously, this could only
be realized in a very cumbersome way: The form configuration must be replaced at runtime via AppCC-coding.

The new functionality allows a far better and more comfortable approach to implement these use-cases.

**BASIC CONCEPTS**

**Adaptation Schema**

The adaptation schema is simply a list of characteristics (or 'dimensions') which can be used for adaptations. In the previous example, there is the need for two characteristics: role and country. The adaptation schema is created using SM30 views `FPM_V_ADAPT_SCHEMA` and `FPM_V_ADAPT_DIM`.

**Adaptation Dimension**

The adaptation dimension represents an individual characteristic within an adaptation schema. It is maintained in view `FPM_V_ADAPT_DIM`. A dimension is defined by a name for identification, an index and a data element. The index is used to determine the dominating adaptation dimension in case there is a collision. The data element is only used for the design-time allowing to you to provide field and F4 help while configuring context-based adaptations.

**Adaptation Context**

The adaptation context is a set of values for a given adaptation schema. So, if the adaptation schema `TEST` consists of the dimensions `ROLE` and `COUNTRY`, a sample adaptation context would be `ROLE = MANAGER` and `COUNTRY = FRANCE`.

The term 'context' in this document is completely independent of the Web Dynpro term 'context'. To avoid misunderstandings, the term 'adaptation context' is used.

**Inheritance of Component Configurations**

Context-based adaptations use the new inheritance concept of component configurations. Each adapted configuration is represented by a derived configuration. A derived configuration contains only the delta to its base configuration.

**STEP-BY-STEP EXAMPLE**

In the first part of this step-by-step example we will extend the existing WD demo application `S_EPM_FPM_PO` by adding an Attachment UIBB to the overview page. This additional UIBB should only be available for Managers. As this can be done without any modification of the existing application, there is no need to create a copy first.

Note that the sections Create the Application Configuration and Create the CBA Configuration can be completed using the CBA Enabler tool (rather than having to do thing manually).

In the second part of this example we will exchange street and house number for the French address format.

**Adding an Attachment UIBB for Managers**

1. Create the Adaptation Schema
As we want to adapt the application with respect to role (only Managers should see the attachment UIBB) and country (French addresses need a different field order), we need to create an adaptation schema having the two dimensions, *Role* and *Country*.

As this adaptation schema already exists (*TEST_FPM*), you can use this and skip the following section.

1. Launch transaction SM30 and enter *FPM_V_ADAPT_SCHM* as Table/View.
2. Choose Create for a new adaptation schema.
3. Edit *FPM_V_ADAPT_DIM* again with SM30 and create the two dimensions needed (see screenshot below).

An adaptation schema should normally be valid throughout a whole application area; there is no need to create an individual schema for every application.

2. Create the WD Application

In order to run *S_EPM_FPM_PO* as an adaptable configuration, a new WD application must be created:

1. Open SE80 and create a new WD application (e.g. *Z_CBA_EPM_FPM_PO*). When you create non-adaptable FPM applications you use one of the *FPM_<FLOORPLAN_ID>_COMPONENT* components for your application. For adaptable applications, you use *FPM_ADAPTABLE_<FLOORPLAN_ID>* instead.
2. Therefore, as *S_EPM_FPM_PO* is an application using the OVP floorplan, you must create the new application using the following settings:
   - Component: *FPM_ADAPTABLE_OVP*
   - Interface View: *FPM_WINDOW*
   - Plug Name: *DEFAULT*
c. To be able to set the adaptation context of your application via URL parameters or via application configuration, add the relevant adaptation dimensions as application parameters (see following screenshot).
3. Create the Application Configuration

As with non-adaptable applications, the next step is to create a configuration for the new application.

Note: The details for creating the application configuration manually are detailed below. You can also use the CBA Enabler tool to do this.

a. On the **Structure** tab of the application configuration editor, you can see that you have to enter two component configuration names (see screenshot):

b. In the first row, you can enter an arbitrary name as this configuration does not yet exist. This configuration will be created shortly and will contain the necessary data for CBA. In the second row, you enter the original application’s root component configuration (the configuration used for the floorplan component `FPM_OVP_COMPONENT`):
4. Create the CBA Configuration

Note: The details for creating the CBA configuration manually are detailed below. You can also use the CBA Enabler tool to do this.

Now create the configuration entered in the first row of the application configuration.

   a. There is only one attribute which has to be maintained within this configuration, the *Adaptation Schema*, under the context node *configData* (See screenshot below). Enter the name of your adaptation schema and save the configuration.

![Component Configuration Screenshot]

Now all the preparation work is done. If you run the new application it should look exactly like the original.

5. Create an Adaptation

   a. Call up your new application configuration again and navigate from there to the OVP component configuration. You should now see a new *Adaptations* toggle-button in the main toolbar; choose it to switch on the adaptation panel.

   b. One entry already exists in the *Adaptations* panel. This is the base configuration which you now can adapt.

   c. To create an adaptation for the *Manager* role, choose the *Add Adaptation* button and enter *MANAGER* as the Role in the dialog box that appears (leave *Country* empty). After entering package details, you will have another entry in the adaptation list. Select it and switch to edit mode.

   d. Now add a new UIBB to the page and save it (see example in the following screenshot):
If you run the application now, it is still unchanged. This is because there is still no adaptation context set. However, the application is now ready to show the additional UIBB when the adaptation context is set to role = MANAGER.

This can now easily be achieved by adding the relevant adaptation context as a URL parameter. In the URL of the application, add &ROLE=MANAGER, refresh the screen; you should now see that the application has been adapted and the attachment UIBB is displayed in the bottom of the overview page.

Before going on to the next part of this step-by-step example, let’s have a quick summary:

We started with a delivered standard application and adapted it. We had to work through a couple of steps but we did not have to enter additional code or modify existing objects; all the steps were purely configurative.

Adapting the Address Layout

In the second part of this example, we will demonstrate some dynamical adjustments of the screen layout based on the data to be displayed. As the chosen application is not yet prepared to support CBA, this second part will require a little bit of coding. As we are using the same application we can skip the steps 1-4 from the previous part and directly continue with preparing the application to support CBA.

1. Extending the Form’s Feeder Class

The data in the original application’s form is provided by the feeder class CL_EPM_PO_FORM_FEEDER. In order to make the form adaptable (with regard to country), the feeder class must communicate the country of the data to the CBA framework.
Creating Context-Based Adaptations with FPM: Step-By-Step Guide

The original feeder class does not do this because it was created before CBA was available: since CBA is new, this applies to all feeder classes at the moment. Therefore, this is quite a common use-case.

To provide the CBA functionality without modifying the standard feeder class, we need a new feeder class. As we do not want to re-implement all its functionality, we subclass it and only redefine the one method which needs to be adapted.

Therefore, open transaction SE24 and create a new class which inherits from CL_FPM_PO_FORM_FEEDER. The only method we redefine is the IF_FPM_GUIBB_FORM~GET_DATA method and we implement it using the following code:

```plaintext
method IF_FPM_GUIBB_FORM~GET_DATA.
* call the GET DATA method of the standard feeder class
CALL METHOD SUPER->IF_FPM_GUIBB_FORM~GET_DATA
EXPORTING
  IO_EVENT = io_event
  IT_SELECTED_FIELDS = IT_SELECTED_FIELDS
  IV_RAISED_BY OWN UI = IV_RAISED_BY OWN UI
  IV_EDIT MODE = IV_EDIT_MODE
IMPORTING
  ET_MESSAGES = ET_MESSAGES
  EV_DATA_CHANGED = EV_DATA_CHANGED
  EV_FIELD_USAGE_CHANGED = EV_FIELD_USAGE_CHANGED
  EV_ACTION_USAGE_CHANGED = EV_ACTION_USAGE_CHANGED
CHANGING
  CS_DATA = CS_DATA
  CT_FIELD_USAGE = CT_FIELD_USAGE
  CT_ACTION_USAGE = CT_ACTION_USAGE.

* Check event id to avoid infinite loop
check io_event->MV_EVENT_ID ne
  IF_FPM_CONSTANTS=>GC_EVENT-ADAPT_CONTEXT.

data: lo_fpm type ref to if_fpm,
  lo_event type ref to CL_FPM_EVENT.
field-symbols: <fs_country_code> type SNWD_COUNTRY.
* determine the country code from the data delivered by the standard feeder
  ASSIGN COMPONENT 'COUNTRY_CODE' of STRUCTURE cs_data to <FS_COUNTRY_CODE>.
* create the object to set the adaptation context.
CREATE OBJECT LO_EVENT
EXPORTING
  IV_EVENT_ID = IF_FPM_CONSTANTS=>GC_EVENT-ADAPT CONTEXT
  IV_ADAPTS контext = ABAP_TRUE.
* Set the adaptation context via event parameters
lo_event->MO_EVENT_DATA->SET_VALUE(exporting iv_key = 'COUNTRY'
  iv_value = <FS_COUNTRY_CODE> ).
* finally raise the event
lo_fpm = CL_FPM_FACTORY=>GET_INSTANCE( ).
lo_fpm->RAISE_EVENT( lo_event ).
endmethod.
```
Let’s now walk through the code step-by-step:

a. First we call the `IF_FPM_GUIBB_FORM~GET_DATA` of the parent class, making sure that the standard logic and data retrieval is executed, so the logic of the feeder class remains unchanged. All we are doing here is adding a few additional lines of code to inform the CBA framework about the changed adaptation context.

b. We then check that the current event is not an adaptation event. This is because setting the adaptation context is done by raising another FPM event, which then calls this method again. Omitting this check would cause an infinite loop.

c. Next we determine the country from the data provided by the standard implementation.

d. Finally we create an event to change the adaptation context, adding the country information as an event parameter and raising the event.

Note that this code is far from optimal; it is kept as simple as possible for example purposes and to demonstrate how CBA works. A better version will be provided in a later chapter.

2. Replacing the Standard Feeder Class

Now we will use a trick to replace the standard feeder class with our newly created one. To do this, create an adaptation of the original form configuration. Call up your new application configuration again. From there, navigate via the OVP component configuration to the Form UIBB configuration. Create a new adaptation for it. As adaptation context set `ROLE = *` and leave the `Country` value empty. Then replace the feeder class within the adapted configuration.

`ROLE = *` means this adaptation should be applied to all adaptation contexts, where no more specific adaptation exists. Since, at the moment, this is the only adaptation of the form, this adaptation will always be applied. Therefore, this is a simple trick to replace a feeder class without modifying any standard object.

These two steps are much easier if you want to prepare your own application for CBA. In that case, you would simply add the necessary lines of code to the standard feeder class application and that’s it; there is no need to create your own feeder class and to replace the feeder class via an adaptation.

Applications which want to provide the adaptation functionality should add the necessary code to their standard code. This way, customers are enabled using CBA completely without these two more complex steps.

3. Creating an Adaptation for French Addresses

The last step is to create an adaptation of the form for the context `COUNTRY = FR`. So, we again navigate to the form’s configuration via the application and the OVP component configuration. Create a new adaptation and set the adaptation context to `ROLE = *` and `COUNTRY = FR`. 
Replace the feeder class and add your own, and swap the street and house number fields (see following screenshot):

Run the application. If you select a French supplier you should now see the adjusted configuration.

Now run your application again, but this time add `ROLE=MANAGER` to the URL. The result is probably no surprise: the additional attachment UIBB is displayed because you started the application as a manager, and for French addresses you get the right order for street and house number.

4. Enhancing the French Address Adaptation

As mentioned already, there is some room for improvement in this example. There are two issues with the current solution:

a. The feeder class is currently firing an additional event to set the adaptation context at every FPM event. This means that the number of FPM round-trips is doubled. Worse still, is if you assume that it should be possible to run the feeder class in an application which does not support CBA, then even in these applications the number of FPM events is doubled. As this might lead to performance implications, this should be avoided.

b. The adaptation context is set globally. This means it will be applied to all UIBBs on the screen. This is normally OK. However, there might be situations where another UIBB on the screen uses another adaptation context.

This is discussed further in the following two sections.
Avoiding Unnecessary FPM Events

To avoid that the CBA-specific code is executed in applications which do not support CBA, you can simply check if \texttt{IF\_FPM\rightarrow MO\_ADAPTATION\_MANAGER} is bound. Add the following line of code directly after the call of the parent’s \texttt{GET\_DATA} method:

\begin{verbatim}
check CL_FPM_FACTORY\rightarrow GET\_INSTANCE( )\rightarrow MO\_ADAPTATION\_MANAGER is bound.
\end{verbatim}

Then, limit the additional FPM events, needed for setting the adaptation context, to those situations where the relevant context has indeed changed; simply store the country value at every event and, in the next event, check if it has changed. Only trigger the adaptation event if there is a change. Add a private attribute \texttt{MV\_COUNTRY\_CODE} (type \texttt{SNWD\_COUNTRY}) to your class and add the following lines of code before you create the event:

\begin{verbatim}
check MV\_COUNTRY\_CODE \neq <FS\_COUNTRY\_CODE>.
MV\_COUNTRY\_CODE = <FS\_COUNTRY\_CODE>.
\end{verbatim}

Setting the Adaptation Context Locally

To set the adaptation context for a single UIBB only, it is possible to set it locally. This can be achieved by adding the UIBB instance key to the event and using an appropriate event.

Let’s start with using an appropriate event. There are two options: Either use \texttt{if\_fpm\_constants\rightarrow gc\_event\_ADAPT\_CONTEXT\_LOCAL} as the event ID or use any other event (except \texttt{if\_fpm\_constants\rightarrow gc\_event\_ADAPT\_CONTEXT}) and set its \texttt{ADAPTS\_CONTEXT} attribute to TRUE. If you then provide the UIBB’s configuration key as an event parameter, \texttt{if\_fpm\_constants\rightarrow gc\_event\_param\_source\_config\_id}, it is automatically treated as a local event.

But, in any case, for a local adaptation change we need the UIBB instance key. A simple solution would be to hardcode this. However, this would then only work for the specified configuration and when the feeder class is reused in a different application, this would not work. Therefore, we should try to determine the instance key at runtime.

This information is passed to the feeder class in the \texttt{IF\_FPM\_GUIBB\_INITIALIZE} method. So again the solution is obvious: Redefine this method, store the UIBB instance information and then use it when firing the event.

Add another private attribute to your feeder: \texttt{MS\_INSTANCE\_KEY type FPM\_S\_CONFIG\_KEY}.

Redefine method \texttt{IF\_FPM\_GUIBB\_INITIALIZE} and add the following code:

\begin{verbatim}
method IF\_FPM\_GUIBB\_INITIALIZE.
    CALL METHOD SUPER\rightarrow IF\_FPM\_GUIBB\_INITIALIZE
        EXPORTING
            IT_PARAMETER = it_parameter
            IO\_APP\_PARAMETER = io_app_parameter
            IV\_COMPONENT\_NAME = iv\_component\_name
            IS\_CONFIG\_KEY = is\_config\_key
            IV\_INSTANCE\_ID = iv\_instance\_id.
        move\_CORRESPONDING IS\_CONFIG\_KEY to MS\_INSTANCE\_KEY.
\end{verbatim}
MS_INSTANCE_KEY-INSTANCE_ID = IV_INSTANCE_ID.

endoDMETHOD.

Trigger the local adaptation event. To do this, enhance the coding of your `GET_DATA` method by firing the `IF_FPM_CONSTANTS=>GC_EVENT-ADAPT_CONTEXT_LOCAL` event instead of the `if_fpm_constants=>gc_event-ADAPT_CONTEXT` event and add the instance key parameters to the event. In the end, the whole `GET_DATA` method should look like this:

```
method IF_FPM_GUIBB_FORM-GET_DATA.
  * call the GET_DATA method of the standard feeder class
  CALL METHOD SUPER=>IF_FPM_GUIBB_FORM-GET_DATA
    EXPORTING
      IO_EVENT = io_event
      IT_SELECTED_FIELDS = IT_SELECTED_FIELDS
      IV_RAISED_BY_OWN_UI = IV_RAISED_BY_OWN_UI
      IV_EDIT_MODE = IV_EDIT_MODE
    IMPORTING
      ET_MESSAGES = ET_MESSAGES
      EV_DATA_CHANGED = EV_DATA_CHANGED
      EV_FIELD_USAGE_CHANGED = EV_FIELD_USAGE_CHANGED
      EV_ACTION_USAGE_CHANGED = EV_ACTION_USAGE_CHANGED
    CHANGING
      CS_DATA = CS_DATA
      CT_FIELD_USAGE = CT_FIELD_USAGE
      CT_ACTION_USAGE = CT_ACTION_USAGE.
  *
  * Only do the CBA processing in case it's active
  check CL_FPM_FACTORY=>GET_INSTANCE( )=>MO_ADAPTATION_MANAGER is bound.
  *
  * Check event id to avoid infinite loop
  check io_event->MV_EVENT_ID ne
    IF_FPM_CONSTANTS=>GC_EVENT-ADAPT_CONTEXT_LOCAL.
  *
  data: lo_fpm type ref to if_fpm,
  lo_event type ref to CL_FPM_EVENT.
  field-symbols: <fs_country_code> type SNWD_COUNTRY.
  * determine the country code from the data delivered by the standard feeder
  ASSIGN COMPONENT 'COUNTRY_CODE' of STRUCTURE cs_data
to <FS_COUNTRY_CODE>.
  *
  * Check that adaptation context needs to be adjusted.
  check MV_COUNTRY_CODE ne <FS_COUNTRY_CODE>.
  MV_COUNTRY_CODE = <FS_COUNTRY_CODE>.
  *
  * create the object to set the adaptation context. The adaptation context
  * is only set locally for the form.
  CREATE OBJECT LO_EVENT
    EXPORTING
      IV_EVENT_ID = IF_FPM_CONSTANTS=>GC_EVENT-ADAPT_CONTEXT_LOCAL
      IV_ADAPTS_CONTEXT = ABAP_TRUE.
  *
  * Set the adaptation context via event parameters
  lo_event->MO_EVENT_DATA->SET_VALUE(
    exporting iv_key = 'COUNTRY'
    iv_value = <FS_COUNTRY_CODE> ).
```
Creating Context-Based Adaptations with FPM: Step-By-Step Guide

* Add the information, for which UIBB the adaptation context is set

```abap
lo_event->MO_EVENT_DATA->SET_VALUE(
  exporting iv_key = if_fpm_constants->source_config_id
  iv_value = MS_INSTANCE_KEY-CONFIG_ID).
lo_event->MO_EVENT_DATA->SET_VALUE(
  exporting iv_key = if_fpm_constants->source_config_type
  iv_value = MS_INSTANCE_KEY-CONFIG_TYPE).
lo_event->MO_EVENT_DATA->SET_VALUE(
  exporting iv_key = if_fpm_constants->source_config_var
  iv_value = MS_INSTANCE_KEY-CONFIG_VAR).
lo_event->MO_EVENT_DATA->SET_VALUE(
  exporting iv_key = if_fpm_constants->SOURCE_INSTANCE_ID
  iv_value = MS_INSTANCE_KEY-INSTANCE_ID).
```

* finally raise the event

```abap
lo_fpm = CL_FPM_FACTORY->GET_INSTANCE( ).
lo_fpm->RAISE_EVENT( lo_event ).
```

endmethod.

If you now run the application, it will still behave as before but now the adaptation context is only applied locally to the Form UIBB.

### Hiding of UIBBs

In this section, we want to prove that the adaptation context is really only applied locally. This gives us the opportunity to demonstrate another feature that CBA offers: dynamically hiding UIBBs based on the adaptation context.

1. Let’s create an adaptation for the List UIBB underneath the form using the same adaptation context and see that it not applied. Navigate again via the application configuration to the List UIBB’s configuration and create an adaptation for `ROLE = * COUNTRY = FR`. In the dialog box where you enter the adaptation context, flag the check box **Only Hide UIBBs**. After closing the dialog box, you are already done. In flagging this checkbox you have stated that this UIBB shall be hidden if the adaptation context fits.

2. Run the application again. It should be unchanged, that is, the List UIBB should still be there. Now let’s try it the other way round. Go to your `GET_DATA` method again and change the code, so that the global adaptation event is thrown (simply delete the `_LOCAL` in the `EVENT ID` constant):

```abap
... CREATE OBJECT LO_EVENT
  EXPORTING
    IV_EVENT_ID = IF_FPM_CONSTANTS->GC_EVENT-
    ADAPTCONTEXT_LOCAL,
    IV_ADAPTS_CONTEXT = ABAP_TRUE.
...```

3. When you execute the application now, the List UIBB should be removed. (Alternatively, you can set a break-point in the first statement after the `CREATE OBJECT LO_EVENT`
statement. Run the application again, select a French supplier and when the break-point is reached, change the value of LO_EVENT->MV_EVENT_ID from FPM_ADAPT_CONTEXT_LOCAL to FPM_ADAPT_CONTEXT. This way, we are changing the event from local to global. Then let the program continue).

Although this is probably a very useful feature, it was only introduced as a work-around for a limitation of CBA.

**Limitation:**
Dynamic changes of the adaptation context at runtime will only affect adaptations of UIBB configurations. The floorplan configurations are loaded at startup of the application and there is no way to replace it later on. Therefore, it is not possible to change the UIBB assembly of a page at runtime via CBA. The only option is hiding of UIBBs.

The natural approach to remove the List GUIBB in the French adaptation context would be to create an adaptation for the floorplan component (as in the first part of this tutorial) for the adaptation context ROLE = *, COUNTRY = FR and remove the List UIBB in this adaptation. But this will not work; the List UIBB will remain on the screen.
© 2014 SAP AG. All rights reserved.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP BusinessObjects Explorer, StreamWork, SAP HANA, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects Software Ltd. Business Objects is an SAP company.

Sybase and Adaptive Server, Anywhere, Sybase 365, SQL Anywhere, and other Sybase products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Sybase Inc. Sybase is an SAP company.

Crossgate, m@tic EDDY, B2B 360°, and B2B 360° Services are registered trademarks of Crossgate AG in Germany and other countries. Crossgate is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

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