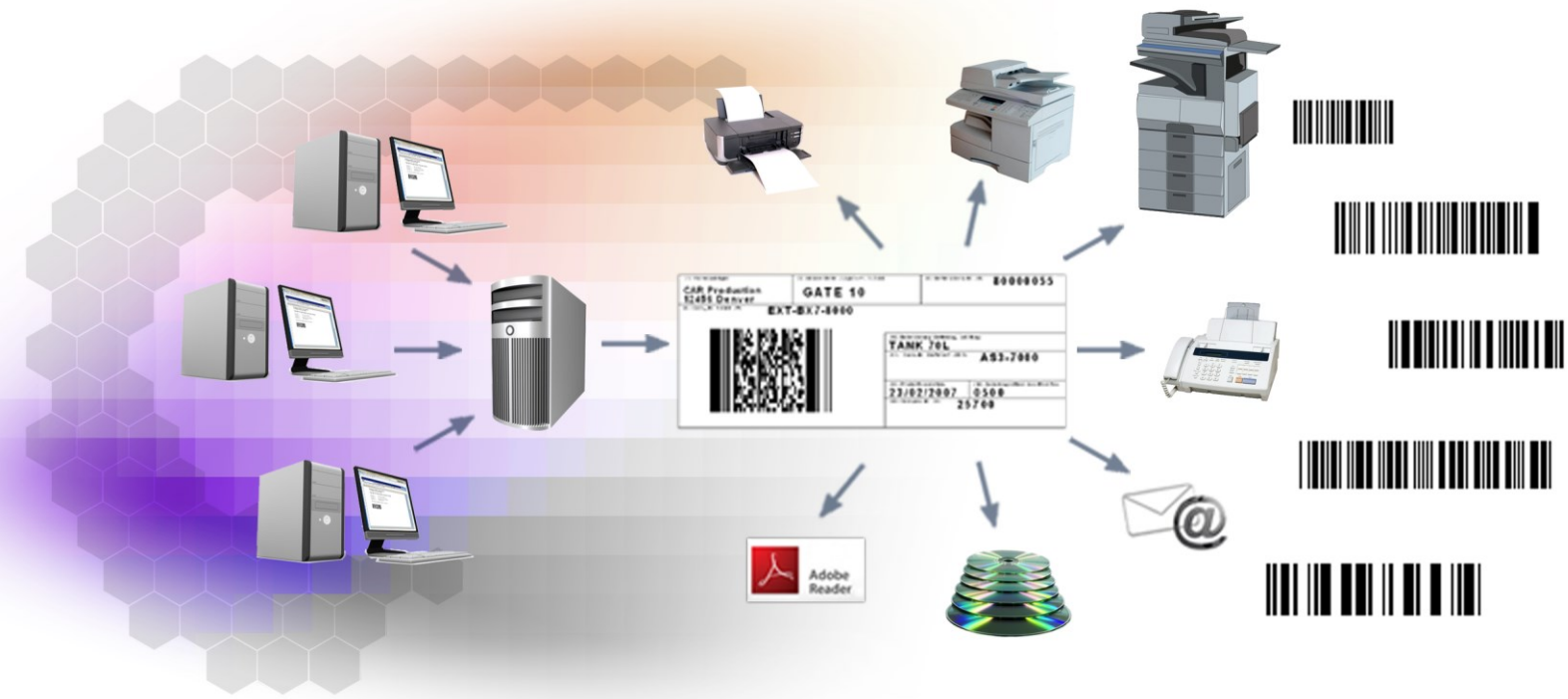
Suchy MIPS

www.suchymips.de



RBarc+

For SAP® Systems

Version 12

Reference Manual

Copyrights and Trademarks

SAP, ABAP and SAPscript are registered trademarks of the SAP AG, Walldorf.
Microsoft Windows is he registered trademark of the Microsoft Corp.

All program names and terms additionally used in this manual are possibly also registered trademarks of certain other manufacturers and must not be used commercially or in any other way. Errors and omissions excluded.

During the compilation of texts and illustrations herein, the greatest possible care was exercised. Errors, however, cannot be entirely excluded. The stated data merely serve as product descriptions and must not be regarded as warranted characteristics in the legal sense. Publishers and authors cannot take over any legal accountability or any liability for erroneous statements and their implications

We at Suchy MIPS will always develop our products further in order to offer you the greatest possible comfort. Therefore, we reserve possible deviations between this manual and the actual product and are asking for your understanding should these occur.

Suchy MIPS
Schichtlstraße 32A
81929 München

TABLE OF CONTENTS

1 Introduction

RBarc+ is an **ABAP** program, serving the **ON-THE-FLY** output of barcodes on documents that were created on SAP systems. The program was designed in such a way that the generation of barcodes is independent of the output type, no matter whether you want **to print, fax, mail or archive** the document carrying the barcode or whether you want to **change it into PDF**. Therefore, in this manual, we will only use the term "**print**" when real printing is actually involved. Otherwise, we will use the term "**output**".

The output of barcodes that were created with RBarc+ can be carried out on any type of printer, provided that a suitable **SAP Device Type** with graphic support is available. The connection type used does not matter in this case, meaning that the output medium can be connected via any chosen SAP connection type.

It is important for you to know that the implementation of a barcode output with RBarc+ does NOT mean any change in the SAP standards. It is only necessary to carry out some adjustments in the SAPscript or SmartForms or AdobeForms forms. The corresponding SAP output programs will undergo no changes. Therefore, all RBarc+ programs are installed in the Z-range of the ABAP-Workbench, so that they will not be involved in any possible system updates.

Below, you will again find a short summary of all important RBarc+ characteristics:

- The output of barcodes will always be true to the original, independent of the output type (printing, faxing, mailing, archiving, and changing into PDF).
- For printers and faxes, no hardware upgrades are necessary.
- The output can be carried out on any type of printer or fax (including inkjet devices).
- The output can be carried out via any connection type.
- The SAP standards will not be changed.
- The installation of RBarc+ is carried out in the Z-range, so that it will not be involved in any system update. .
- The installation will only be carried out on the ABAP Application Server (no client adjustment is necessary).
- RBarc+ is operating independently of the operating system on which your SAP system was installed.

This manual is addressed to system administrators (installation) as well as to software engineers and programmers (implementation of barcodes in forms). For the installation, we are assuming a basic knowledge of the ABAP Workbench as well as a basic knowledge of ABAP and the utilized form-composer (SAPscript or SmartForms or AdobeForms). Please understand that we cannot go into exact detail here, such as for example, how to operate the ABAP Workbench. That would reach beyond the scope of this manual. However, please feel free to contact our Support via info@suchymips.de, should you have specific questions.

Note: User, who print documents including barcodes generated by RBarc+ must have following permissions for the object **S_BDS_DS:**

ACTVT = 01,02,03 and 06
CLASSNAME = DEVC_STXD_BITMAP
CLASSTYPE = OT

The appropriate settings may be performed with the transaction "**PFCG**".
The User Role must include the transaction **SE78** and the Authorisation for **BC-SRV-KPR-BDS** (Technical Name **S_BDS_DS**).**S_BDS_DS** is assigned to class "**Basis-Central functions**".
If the permission is missing, no barcode will appear on the output or generated barcodes will not be deleted from the system.

2 Installation

RBarc+ consists of interconnected ABAP programs of the type “Object Program” and „Include“. The programs have to be installed once on the ABAP Application Server. After installation, the form routines implemented in RBarc+ are fully available and can be called from SAPscript as well as from SmartForms forms. If the Demo version of RBarc is already installed on your SAP test system, it is enough to exchange the content of programs (reports) and includes. But the TrueType fonts (Page 6) and the test material (Page 8) have to be newly installed.

Proceed as follows in order to install **RBarc** on your SAP systems. During installation, please pay regard to and strictly observe the described installation sequence as well as the exact naming of the objects:

① Create a New Package:

- Start the transaction **SE80** and create a new package (our suggestion: **ZRBARC_12**, in which the delivered programs and test forms can be installed. Also, create a new order, so that later you will be able to transfer your installation to the target system without any problems. Although for the package you can choose any name beginning with „**Z**“ or „**Y**“, we will in the further course of this manual always refer to the suggested name **ZRBARC_12**.

② Include ZRBARC_12:

- In the package **ZRBARC**, create an object of the type **Include** and name it **ZRBARC_12**.
- Copy the contents from the file **ZRBARC_12.INC** into the source code window of the newly created object.
- Save and **activate** the program.

③ Include ZSS_12 (for SAPscript support):

- In the package **ZRBARC**, create an object of the type **Include** and name it **ZSS_12**.
- Copy the contents from the file **ZSS_12.INC** into the source code window of the newly created object.
- Save and **activate** the program.

④ Include ZSF_12 (for SmartForms support):

- In the package **ZRBARC_12**, create an object of the type **Include** and name it **ZSF_12**.
- Copy the contents from the file **ZSF_12.INC** into the source code window of the newly created object.
- Save and **activate** the program.

③ Include ZAF_12 (for AdobeForms-Support):

- In the packet **ZRBARC_12,** create an object of the type **Include** and call it **ZAF_12**.
- Copy the contents of file **ZAF_12.INC** into the source code window of the newly created object.
- Save and **activate** the programme.

⑤ Program ZSS_BC_SETTINGS12 (for SAPScript support):

- In the package **ZRBARC**, create an object of the type **Program** and name it **ZSS_BC_SETTINGS12.**
- Copy the contents from the file **ZSS_BC_SETTINGS12.PRG** into the source code window of the newly object.

- Save and **activate** the program.

⑥ Program ZSF_BC_SETTINGS12 (for SmartForms support):

- In the package **ZRBARC_12** create an object of the type **Program** and name it **ZSF_BC_SETTINGS12.**

- Copy the contents from the file **ZSF_BC_SETTINGS12.PRG** into the source code window of the newly created object.

- Save and **activate** the program.

⑦ Programm ZAF_BC_SETTINGS (for AdobeForms-Support):

- In the packet **ZRBARC_12,** create an object of the type **Programm** and call it **ZAF_BC_SETTINGS12.**

- Copy the contents of file **ZAF_BC_SETTINGS.PRG** into the source code window of the newly created object

- Save and **activate** the programme.

⑦ TrueType Fonts for a Vertical Output of the Human Readable Text (HRT)

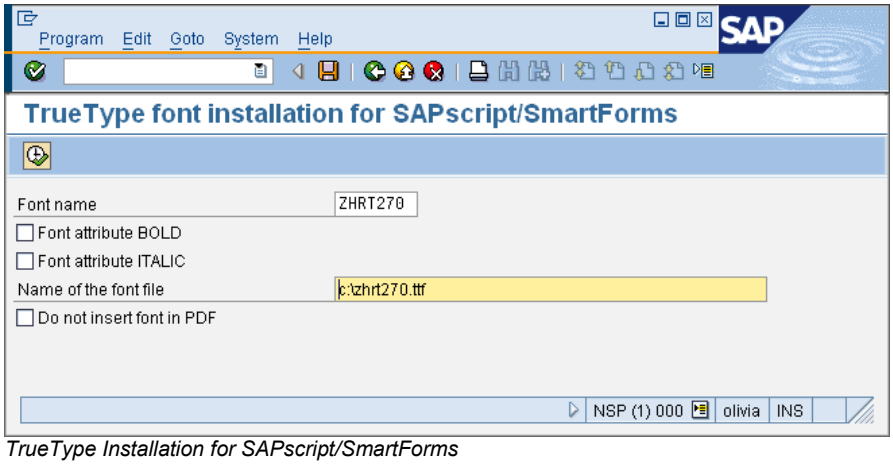
Neither SAPscript nor SmartForms offer the possibility for an output of dynamic text vertically or rotated by 180 degrees**.** Therefore, specific measures are necessary should you wish to rotate a barcode and at the same time output the HRT fitting that barcode, as shows the example illustrated below.

Barcode rotated by 270 degrees

However, in order to offer you the opportunity for such an output, we are delivering 3 True Type fonts together with RBarc+ 3, with the single fonts already rotated by 90, 270 or 180 degrees. When you install these fonts on your SAP system and use them to format your HRT, you will achieve the desired result. The exact procedure will be described in detail further on in this manual, in the chapters 5.3 (for SAPscript) and 6.3 (for SmartForms).

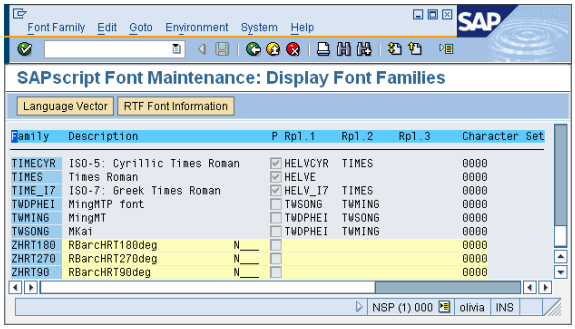
- Together with **RBarc**, the following TrueType fonts are delivered: **ZHRT90.TTF, ZHRT180.TTF** and **ZHRT270.TTF**. In order to install these fonts, please start the transaction **SE73** and select *Install TrueType Font*.

- Enter the relevant font name into the field *Fontname* (without the extension .TTF) and carry out the program.



TrueType Installation for SAPscript/SmartForms

- Following the installation with transaction **SE73**, please check whether the font families **ZHRT90**, **ZHRT180** and **ZHRT270** were actually created. The result should approximately present itself as follows:



List of the installed Font Families

Note: Should you use the device type SAPWIN, printing via the SAP Client Program SAPLPD, you will also have to install the fonts on the Microsoft Windows Clients, otherwise the output of rotated HRT cannot be carried out correctly.

In addition to the programs that are necessary for the output of barcodes, we are providing you with some **test resources** which you can use to test the barcode output immediately after installation – that way you will not have to carry out a previous barcode implementation in your forms. In order to install these resources, please proceede as follows:

① Program ZSS_BC_PRINT (for printing an SAPscript form):

- In the package **ZRBARC_12**, create an object of the type **Program** and name it **ZSS_BC_PRINT.**
- Copy the contents from the file **ZSS_BC_PRINT.PRG** into the source code window of the newly created object.
- Save and **activate** the program.

② SAPscript Form ZSS_BC_FORM:

- Start the ABAP Editor (transaction **SE38**) and then the SAP program **RSTXSCRP.**
- In the field *Objectname*, please enter the form name **ZSS_BC_FORM** and in the field *Mode* the value **IMPORT**, and then activate the program. From the file selection menu, please select the file **ZSS_BC_FORM**.**FOR** and click on **<OK>** in order to terminate the procedure.

③ SmartForms Form ZSF_BC_FORM:

- Start the transaction **SMARTFORMS**.
- In the field *Form*, please enter the name of the form **ZSF_BC_FORM**.
- From the menu, select *Tools / Upload*. From the file selection menu, select the file **ZSF_BC_FORM.XML** and click on **<OK>** in order to terminate the procedure.

④ SmartForms Style ZSF_BC_STIL (full version only):

- Start the transaction **SMARTFORMS**.
- In the field *Style,* please enter the name of the style **ZSF_BC_STIL**.
- From the menu, select *Tools / Upload Style*. From the file selection menu, select the file **ZSF_BC_STIL.XML** and click on **<OK>** in order to terminate the procedure.

⑤ Programm ZAF_BC_PRINT (for printing AdobeForms Forms):

- In the packet **ZRBARC_12**, create an object if the type **Programm** and call it **ZAF_BC_PRINT.**
- Copy the contents of the file **ZAF_BC_PRINT.PRG** into the source code window of the newly created object.
- Save and **activate** the programme.

⑥ AdobeForms Interface ZAF_BC_INTERFACE:

- Start transaction **SFP**
- In the field *Interface* enter the name of the interface **ZAF_BC_INTERFACE**.
- Select the menu item *Tool / Upload Formobject*. From the file menu select file **ZAF_BC_INTERFACE.XML** and click on **<OK>** to end the process.

⑦ AdobeForms Form ZAF_BC_FORM:

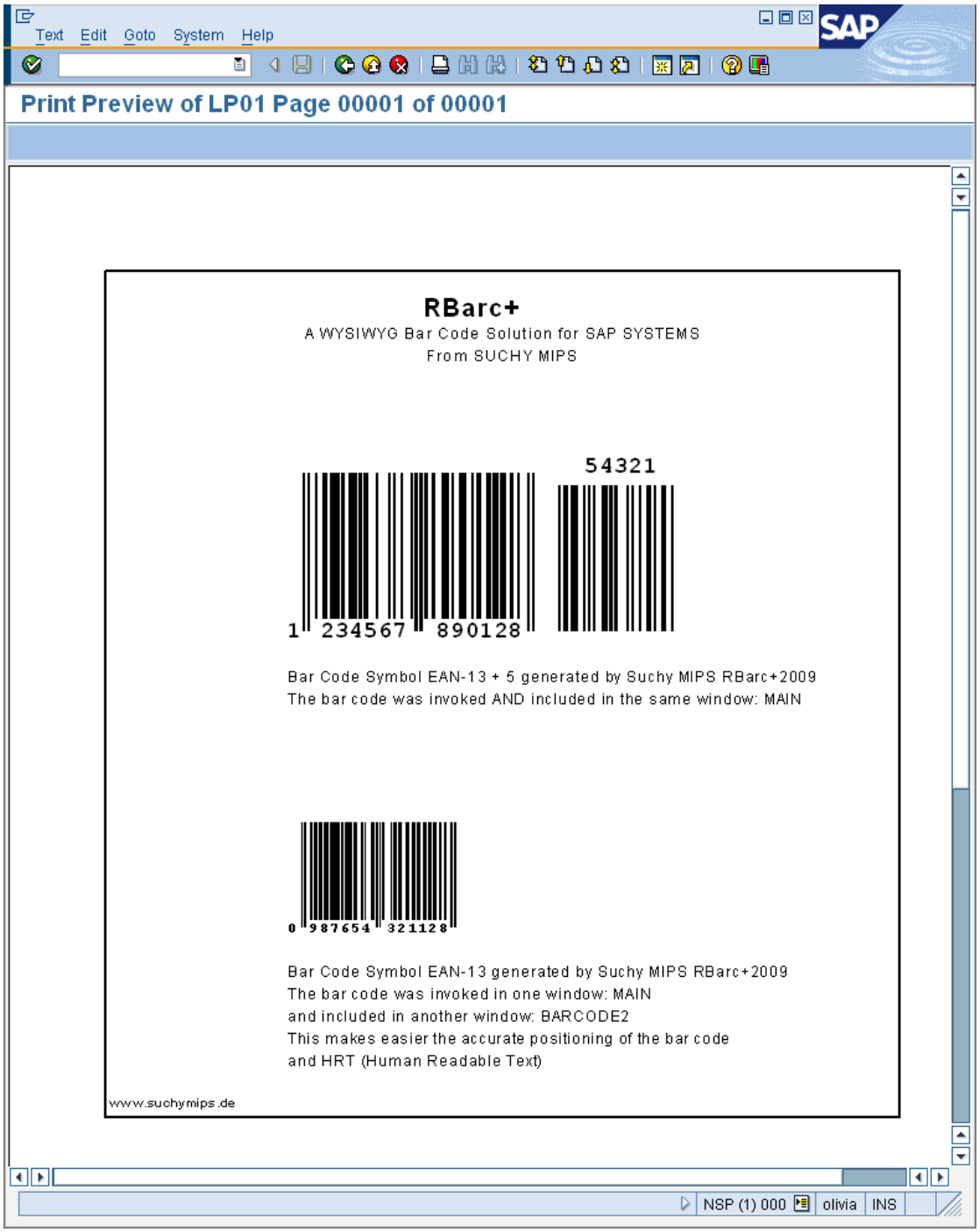
- Start transaction **SFP**
- In the field *Form* enter the name of the form **ZAF_BC_FORM**.
- Select the menu item *Tool / Upload Formobject*. From the file menu select file **ZAF_BC_FORM.XML** and click on **<OK>** to end the process.

Should you have closely followed all the steps described in Chapter 2, the installation is now completed. You are now able to turn your attention to the implementation of barcodes in your forms. In this context, please read the following chapters.

3 Test Print with SAPscript

- Start the ABAP Editor with transaction **SE38** and activate the program **ZSS_BC_PRINT**.

The program **ZSS_BC_PRINT** prints the **SAPscript** form included in the delivery: **ZSS_BC_FORM**. As the barcode output with **RBarc** is device-independent, you do not have to actually print the form, you can just view the results on your screen by selecting *Print Preview* from your printer dialog. The result should approximately present itself as follows:



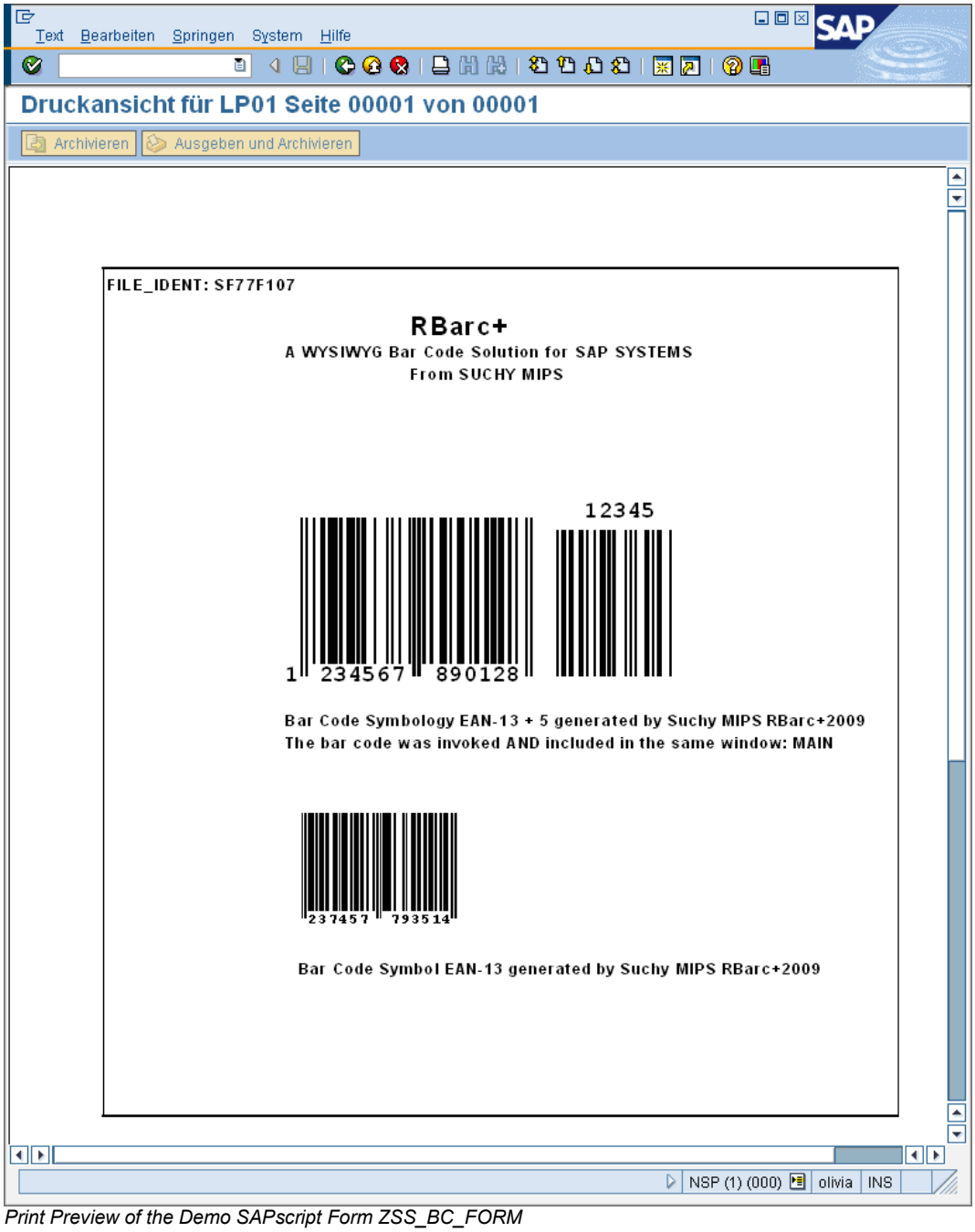
Print Preview of the Demo SAPscript Form ZSS_BC_FORM

The fact that you can see the barcodes on the form output means the installation was successful. Now, you can begin using RBarc+ to implement barcodes in your own SAPscript forms.

4 Test Print with SmartForms

- Start the transaction **SMARTFORMS**.
- In the field *Form*, enter the name of the form **ZSF_BC_FORM** and click on the symbol  or select *SmartForms / Test* from the menu.

As the barcode output with **RBarc** is device-independent, you do not have to actually print the form, you can just view the results on your screen by selecting *Print Preview* from your printer dialog. The result should approximately present itself as follows:



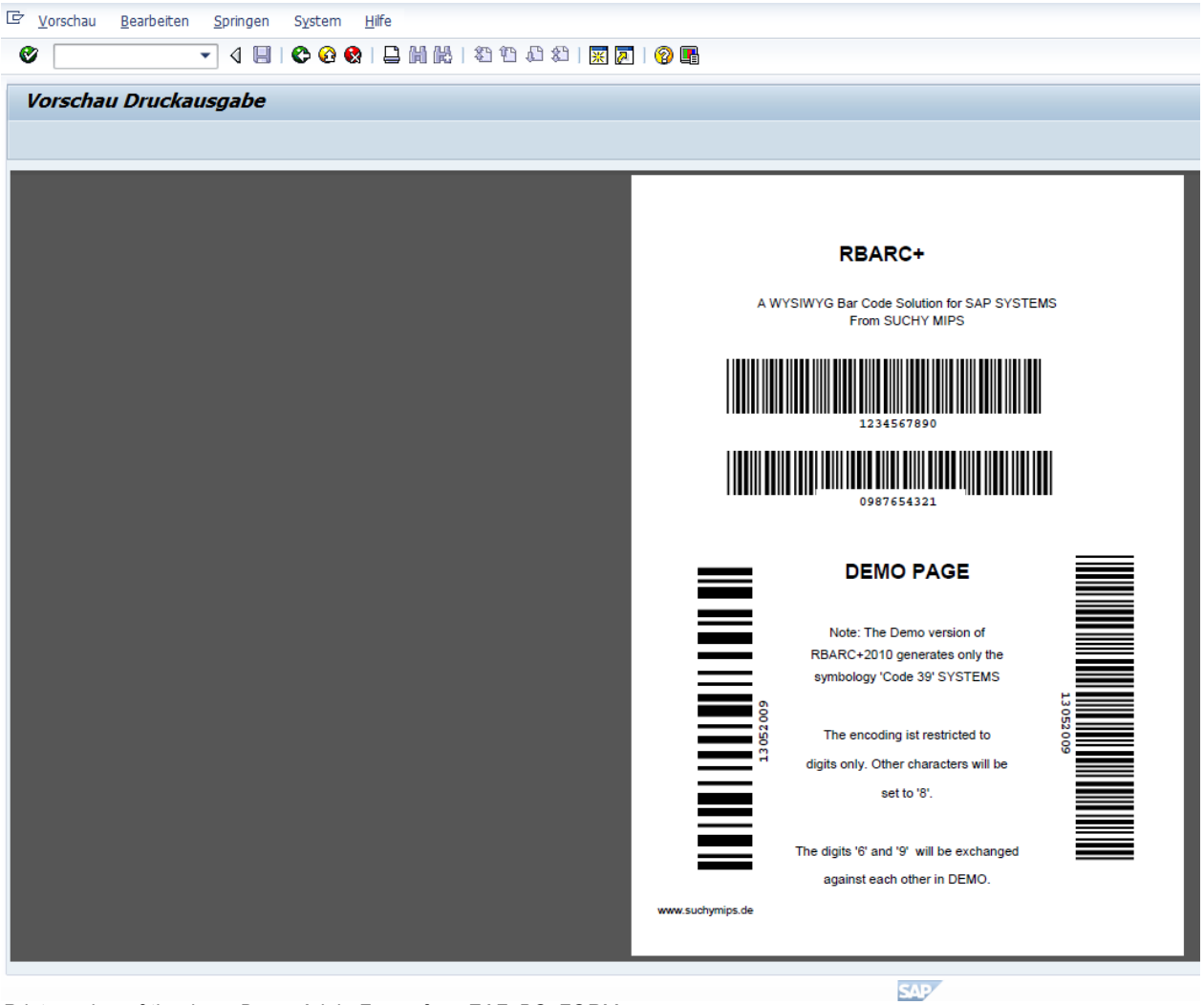
Print Preview of the Demo SAPscript Form ZSS_BC_FORM

The fact that you can see the barcodes on the form output means the installation was successful. Now, you can begin using RBarc+ to implement barcodes in your own SmartForms forms.

5 Test Print with AdobeForms

- Start the ABAP Editor with transaction **SE38** and run the program **ZAF_BC_PRINT**.

The program **ZAF_BC_PRINT** prints the delivered **AdobeForms-**form **ZAF_BC_FORM**. As the barcode output with **RBarc+** is device-independent, there is no need to print the form, you can view the result on the screen by selecting **Print View** from the printer dialog. The result should more or less look as follows:



Print preview of the demo Demo AdobeForms form ZAF_BC_FORM

If you can view the barcodes on the form output, that means the installation was successful. You can now begin to implement barcodes in your own AdobeForms-Forms with RBarc+.

6 Implementation of a Barcode Output with SAPscript

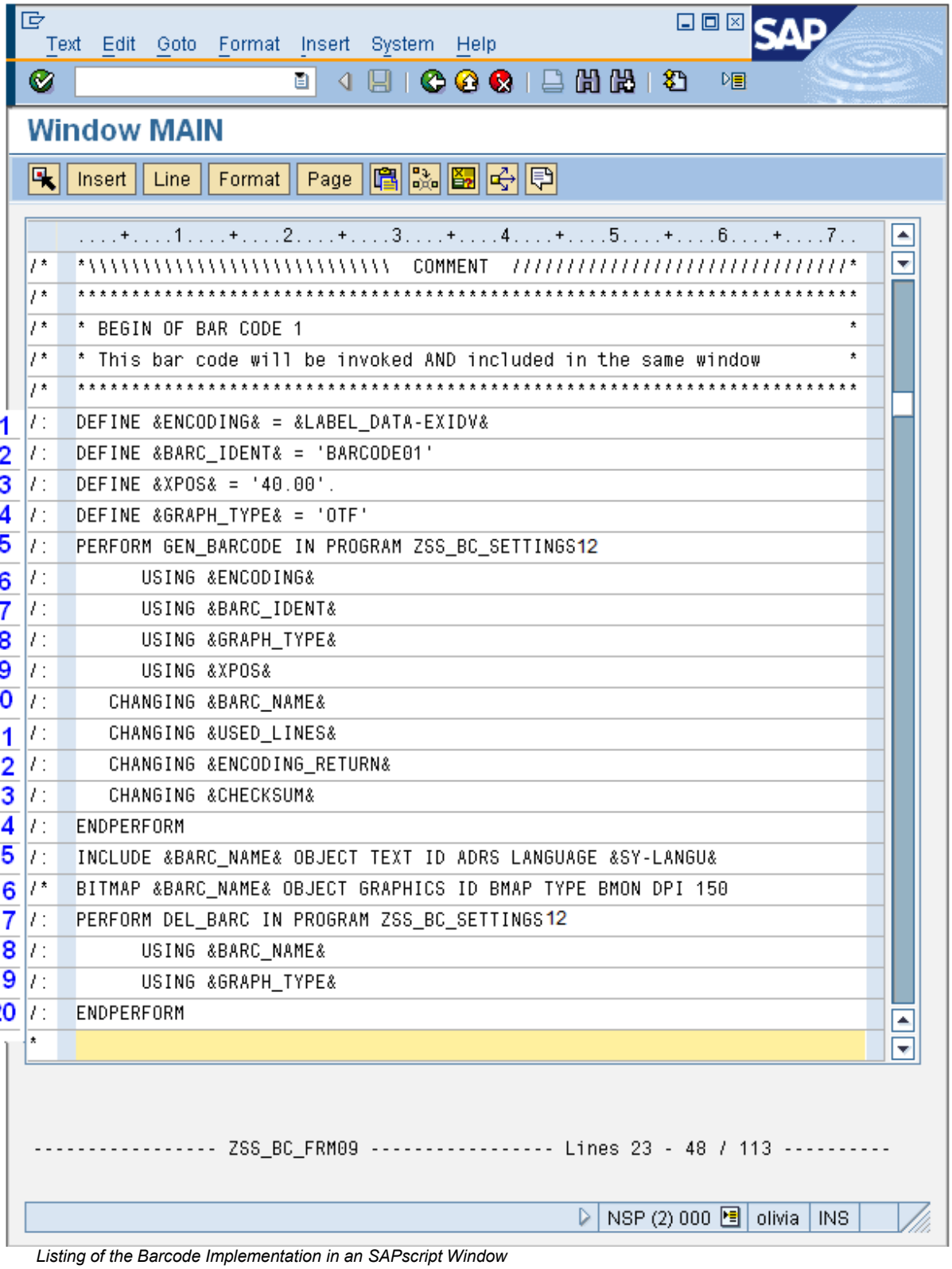
6.1 Embedding Barcodes into an SAPscript Form

Any amount of barcodes can be embedded into an SAPscript form. Although it is advisable to define a separate window for each barcode, a barcode can also be embedded into any existing window. In order to simplify matters, our description concentrates on the embedding of a barcode into an existing SAPscript window. The barcode embedding is solely carried out via so-called command-instructions so that the form layout cannot change (the inserted barcode does not use any of its own space). Existing printing programs for an SAPscript form will **NOT** be affected. The adjustments are only concerning the form itself (that way, the RBarc+ solution is SAP-updatable). To make sure that the form adjustments remain as marginal as possible, only the most necessary definitions are carried out. Among these is the information „**what needs to be encoded**"? and „**where should the barcode be positioned**"? All other barcode properties, such as **symbology**, **width**, **height,** etc. are conducted in the separate ABAP program **ZSS_BC_SETTINGS12**.

The procedure logics are always the same and conform to the following principle:

- One-time definition of the barcode properties (symbology, height, width, etc.) in a separate form routine in the ABAP program **ZSS_BC_SETTINGS12**.
- Definition of the variable(s) in an SAPscript form to be encoded as a barcode.
- Determination of a barcode identification (each barcode within a form has to be clearly identifiable).
- Calling up the form routine **GEN_BARCODE** in program **ZSS_BC_SETTINGS12** and transfer of the parameters (among which are chiefly the data to be encoded and the barcode identification).
- Transfer of the return parameters coming back from RBarc+ (chiefly the dynamic name of the barcode graphic to be embedded).
- Dynamic embedding of the barcode graphic into the form.
- Deletion of the previously created barcode graphic from the system.

We will later hear more about the barcode properties, such as **symbology**, **height**, **width,** etc., as this part is identical for **SAPscript** and **SmartForms** applications. In the following, we will present a listing of an **SAPscript** form with the corresponding explanations, which should enable you to use RBarc+ to embed your own barcodes into **SAPscript** forms.



Listing of the Barcode Implementation in an SAPscript Window

As mentioned before, all lines are command lines that do not alter the form layout, as the barcode graphic does not take up any of its own space. An explanation concerning these listings can be found on the following pages.

Explanation of the Single Lines:

Line 1:

DEFINE &ENCODING& = &LABEL_DATA-EXIDV&

Here, the value of variable **EXIDV** from table **LABEL_DATA**, (which, here, belongs to the setup logic of the SAPscript form) is allocated to variable **ENCODING**. Variable **ENCODING** transfers the value to be encoded to RBac+. The name „**ENCODING**“, therefore, must not be changed. Only one **ENCODING** variable per RBarc+ callup can be transferred. If necessary, however, several different variables from the form procedure logic can be written into one **ENCODING** variable at the same time. In that case, a colon has to be put before the equal sign and the variables have to be put in inverted commas, e.g.:

DEFINE &ENCODING& **:= '**&LABEL_DATA-EXIDV&&LABEL_DATA-EXIDT&**'**

Note: The length of variable **ENCODING** must not exceed altogether **70 characters.** .

Line 2:

DEFINE &BARC_IDENT& = 'BARCODE01'

Here, a barcode identification is allocated that is transferred to RBarc+ via variable **BARC_IDENT**. The maximum length for this variable is **10 characters**. Each barcode in the form has to possess its own, clear identification. Therefore, we recommend using a counter as a means of discrimination, like „01“ in the above example.

Note: The barcode identification has to correspond to the name of the form routine in program **ZSS_BC_SETTINGS12**, in which the barcode properties are defined. Where required, you will perhaps have to create your own form routine for a new barcode in program **ZSS_BC_SETTINGS12**. Our delivery comprises 4 pre-defined form routines in program **ZSS_BC_SETTINGS12**. Those are **BARCODE01, BARCODE02, BARCODE03** and **BARCODE04.** The barcode parameters defined therein can be changed at any time. Chapter 7 will offer you a detailed description.

Line 3:

DEFINE &XPOS& = '40.00'.

With this definition, the barcode is moved to the right by 40 units. The measuring unit for the value corresponds to that of barcode measurements, as they are determined in program **ZSS_BC_SETTINGS12** in the relevant form routine in which barcode properties were defined (parameter **UNIT**). The standard value is „mm“.

Note: With parameter **XPOS,** a barcode can only be moved to the right. Moving it to the left (i.e. beyond the left window frame) is not possible.

Note: It is not possible to move a barcode with the similar parameter **YPOS** up or down, relative to the current cursor position. In principle, entering **YPOS** parameters is possible. This, however, would ultimately lead to the effect that the barcode appears not in the current window but always at the same height of the physical page.

Line 4:

DEFINE &GRAPH_TYPE& = 'OTF'.

With this definition, the kind of graphic type is determined, that is created by RBarc+. As a standard, **OTF** is created (if possible, this parameter should not be changed). In principle, it is possible to create a so-called BMP-Bitmap (DEFINE &GRAPH_TYPE& = 'BMP'). As this cannot be moved to the right relative to the window, this graphic type is almost never used under SAPscript.

Lines 5-9:

```
PERFORM GEN_BARCODE IN PROGRAM ZSS_BC_SETTINGS12.
    USING &ENCODING&
    USING &BARC_IDENT&
    USING &GRAPH_TYPE&
    USING &XPOS&
```

Calling up the form routine **GEN_BARCODE** in program **ZSS_BC_SETTINGS12** and transfer of the parameters **ENCODING**, **BARC_IDENT**, **GRAPH_TYPE** and **XPOS**.

Note: In principle, it is possible to use a separate ABAP program for each form with the corresponding barcode properties defined therein. In that case, please copy program **ZSS_BC_SETTINGS12** into a different program and then call this up from the SAPscript form. When you do this, please make sue that the names of the variables are not changed, as otherwise the program could not function properly.

Lines 10-13:

```
CHANGING &BARC_NAME&
CHANGING &USED_LINES&
CHANGING &ENCODING_RETURN&
CHANGING &CHECKSUM&
```

In the rows 10-13, variables for the return values from Rbarc+ are defined. These names must not be changed. Should you require these variables for other purposes, it is recommended to write these into different variables that you defined yourself.

BARC_NAME – name of the graphic created by RBarc+ with the barcode image.
USED_LINES – defines how many lines the barcode uses (at 1/6th of an inch).
ENCODING_RETURN – defines everything that was actually encoded.
CHECKSUM – the check digit calculated by RBarc (if required).

Note: The returned value for **ENCODING_RETURN** can sometimes deviate from the transferred value for **ENCODING**, e.g. when certain options, such as the deletion of leading zeros, were pre-set in program **ZSS_BC_SETTINGS12**.

.

Line 14:

ENDPERFORM

End of the PERFORM command initiated in line 5.

Line 15:

INCLUDE &BARC_NAME& OBJECT TEXT ID ADRS LANGUAGE &SY-LANGU&

With this command, the barcode graphic created by RBarc+ will be embedded into the form dynamically, i.e. during runtime. This command only applies to **OTF** graphics (recommended standard).

Line 16:

BITMAP &BARC_NAME& OBJECT GRAPHICS ID BMAP TYPE BMON DPI 150

This command is commented out in the form and, here, serves only documentary purposes. Should you have decided to use the graphic type **BMP** (by entering DEFINE &GRAPH_TYPE& = 'BMP' in line 4), then the graphic has to be dynamically embedded with this command.

Note: In contrast to OTF-graphics, with BMP-graphics the resolution has to be defined in DPI. This resolution value has to correspond strictly to the one defined in program **ZSS_BC_SETTINGS12** for the barcode to be inserted. Should that not be the case, a different barcode size than expected will be displayed.

Note: A graphic of the type **BMP**, can – in contrast to an **OTF**-graphic – not be moved horizontally within the window. The parameter **XPOS** remains without effect. For reasons of compatibility with earlier RBarc+ versions that had no **OTF**-formats, the **BMP** format is supported.

Lines 17-20:

```
PERFORM DEL_BARC IN PROGRAM ZSS_BC_SETTINGS12
    USING &BARC_NAME&
    USING &GRAPH_TYPE&
ENDPERFORM
```

With these lines, the barcode that was previously created is deleted from the system.

6.2 Insertion of a Human Readable Text (HRT)

For technical reasons, the HRT cannot be created together with the barcode graphic. Should an HRT be required, it therefore has to be separately output in the form. For these cases, there are two possibilities on offer:

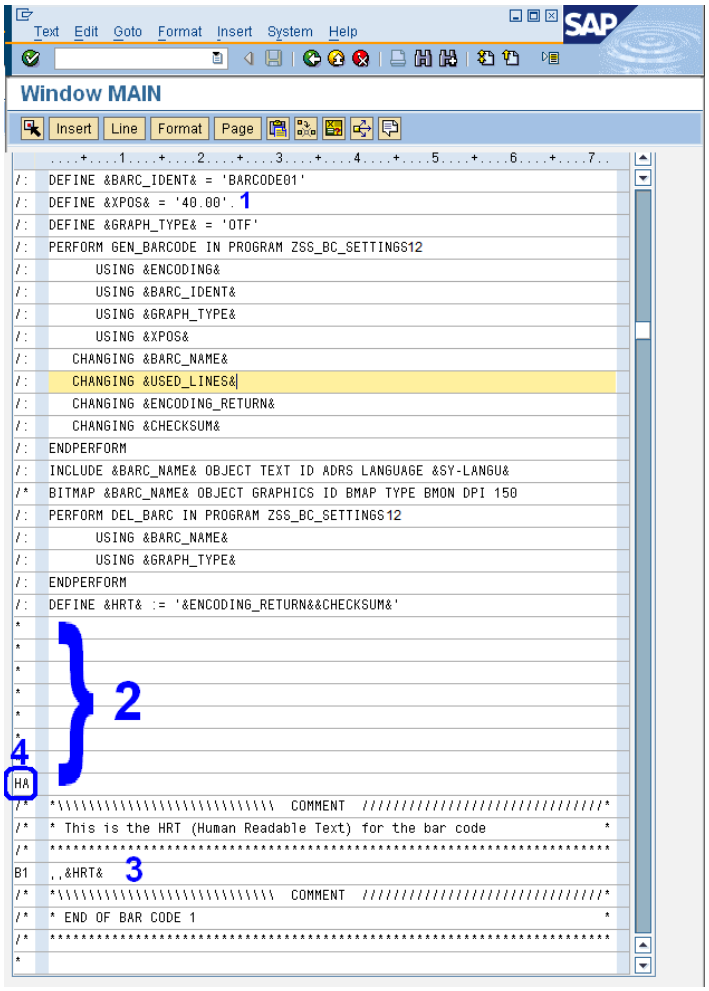
- Define the HRT in the same window as the barcode.
- Define the HRT in its own, separate window.

To start with, we will follow the first approach:

For the HRT, the value of variable **ENCODING_RETURN** should be used. Should the check digit be output together with the HRT, the variables have to be bundled, e.g. in an SAPscript command line with the following contents:

DEFINE &HRT& **:= '**&ENCODING_RETURN&&CHECKSUM&**'**

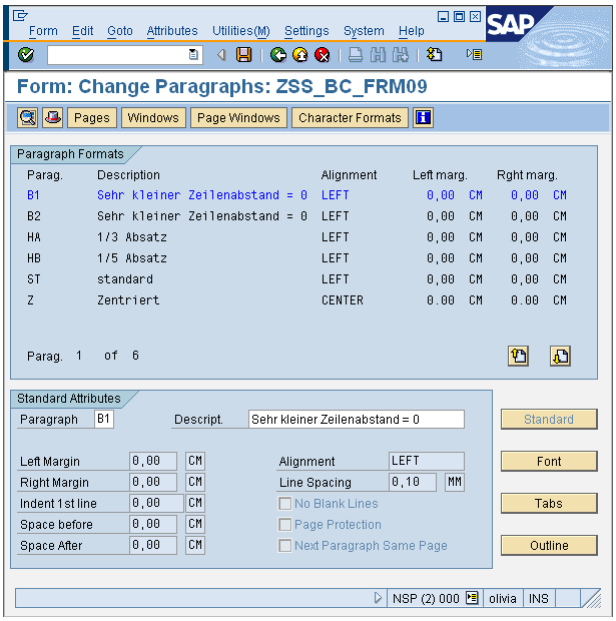
Should you wish to insert the HRT into the same window as the barcode, you have to keep in mind that the barcode does not take up any of its own space. Therefore, you will have to insert the barcode a few lines further down, after the include command for the barcode graphic, in order to position the HRT below the barcode. The horizontal position of the HRT, you can manipulate by using spaces or tab stops before variable **&HRT&**:



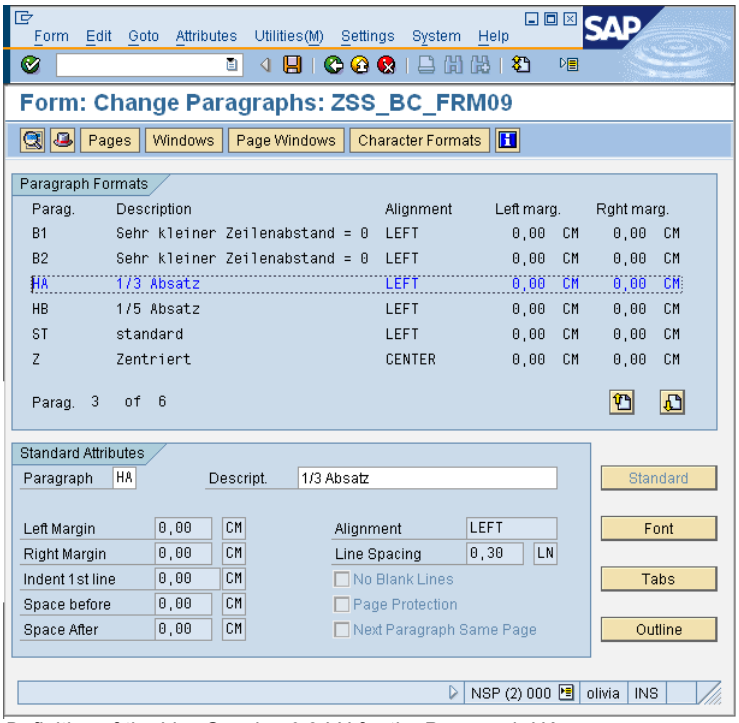
Barcode Implementation in the SAPscript Window with HRT

In order to exactly arrive at the position under the barcode, spaces were inserted under the command lines for the barcode implementation (2). Only then, the HRT can follow (3). As the barcode was moved to the right by 40 mm (1), a tab stop was defined for paragraph „**B1**“ on position 42mm and a tabulator was set before variable **HRT** (3).

It is not always possible to reach the exact vertical position for the HRT by using standard lines (1/6 inch), you will probably have to define an additional paragraph format with its separately defined line spacing. In the above example, a paragraph „HA" was inserted **(4)**, defined with a line spacing of 1/3 LN.

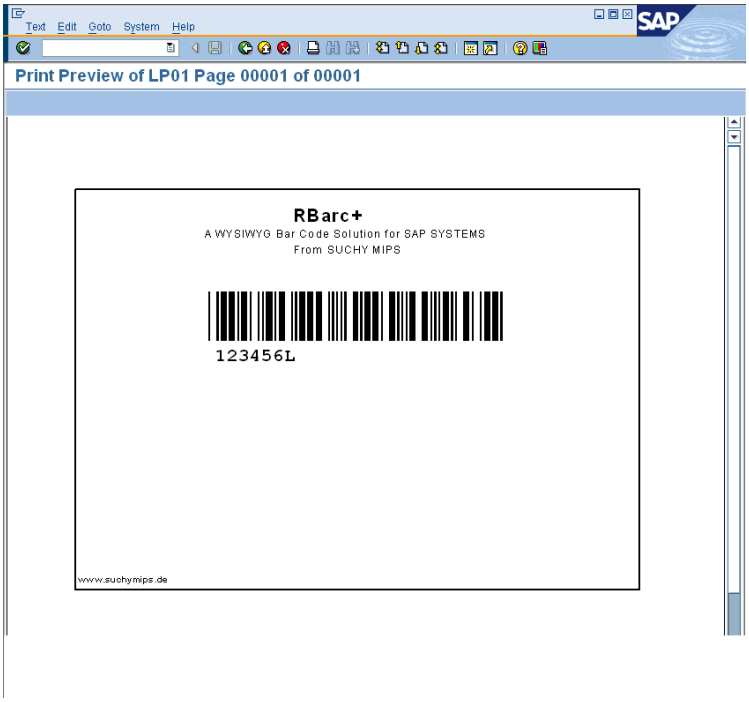


Tabulator Definition for the Paragraph „B1".



Definition of the Line Spacing 0.3 LN for the Paragraph HA.

The result will present itself approximately as follows:



By using the methods described above, it is possible to output the **HRT** at any desired position.

6.3 Insertion of a Rotated Human Readable Text (HRT) into an SAPscript Form

As already mentioned, neither SAPscript nor SmartForms include the possibility to rotate a dynamic text. Therefore, we have developed three special fonts for you with the characters already rotated by 90, 180 or 270 degrees. If you have carried out all the single installation steps, these fonts should already be installed on your SAP-System. If that is not the case, please return to the installation instructions in Chapter 3 and install the TrueType fonts **ZHRT90**, **ZHRT180** and **ZHRT270**, which were included in the delivery.

Example HRT90:
1 2 3 4 5

Example HRT180
12345

Example HRT270
1 2 3 4 5

As the character feed is carried out in horizontal direction in spite of the fonts having been rotated, each character of the HRT has to be output in a separate line for a vertically oriented text:

1
2
3
4
5

As you will certainly already have noticed, in some cases the sequence of the figures is shown reversely to the original character sequence 12345. Therefore, if you wish to output text that was rotated by 90 or 180 degrees, you will have to output the characters of the HRT in reversed order. In this case, SAPscript offers the corresponding possibilities described in the following:
Let us assume that the HRT is 10 characters long and was saved in variable **HRT1.** Then, in case of a text rotated by 90 or 180 – seen from above or from the left – first the tenth, then the ninth, then the eighth, etc. character has to be output. You can achieve this using the following SAPscript encoding:

Example HRT1 = '0123456789'

Command Lines for 90 degrees	**Result with a 90-degree rotation**
&HRT1+9(1)&	9
&HRT1+8(1)&	8
&HRT1+7(1)&	7
&HRT1+6(1)&	6
&HRT1+5(1)&	6
&HRT1+4(1)&	5
&HRT1+3(1)&	4
&HRT1+2(1)&	3
&HRT1+1(1)&	2
&HRT1(1)&	1

Command Line

```
&HRT1+9(1)&&HRT1+8(1)&&HRT1+7(1)&&HRT1+6(1)&&HRT1+5(1)&&HRT1+4(1)&&HRT1+3(1)&&HRT1+2(1)&HRT1+1(1)&&HRT1(1)&
```

Result with a 180-degree rotation

```
9876543210
```

In case of a 270-degree rotation, the command sequence has to be in reverse to the 90-degree rotation:

Command Lines for 120 degrees	**Result with a 270-degree rotation**
&HRT1+(1)&	0
&HRT1+1(1)&	1
&HRT1+2(1)&	2
&HRT1+3(1)&	3
&HRT1+4(1)&	4
&HRT1+5(1)&	5
&HRT1+6(1)&	6
&HRT1+7(1)&	7
&HRT1+8(1)&	8
&HRT1+9(1)&	9

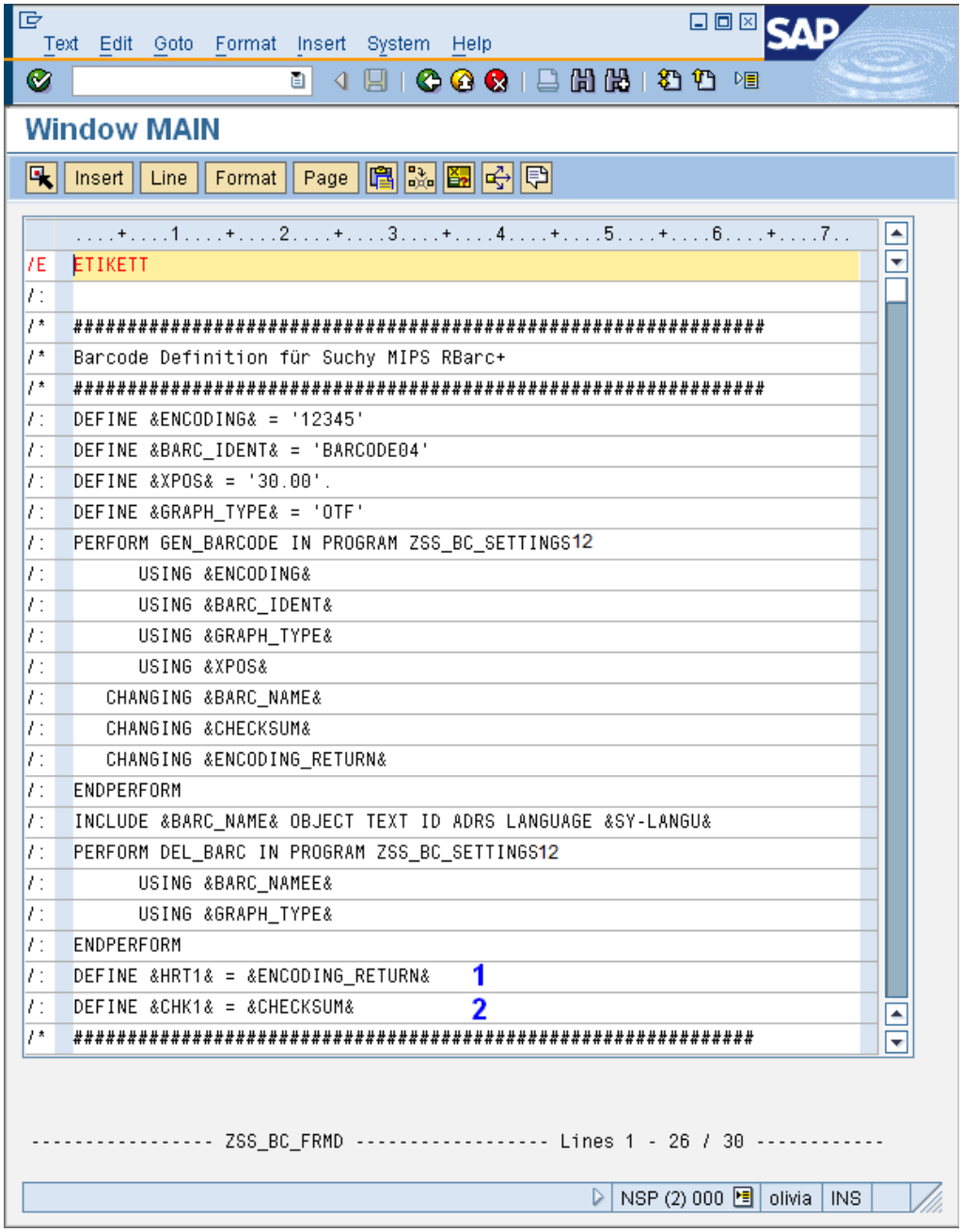
Note: Should it, in case of a 90 or 270-degree rotation, be necessary to set the characters closer or wider apart, you will have to define a corresponding paragraph in the SAPscript form and allocate this to the single lines of the HRT output in the command column.

As it is often difficult to lodge this coding inside an existing SAPscript window without tampering with the layout, we recommend to use a separate window for the HRT. That has the additional advantage that the correct positioning of the HRT can be comfortably achieved via the window position.

The following example of an SAPscript form with a barcode rotated by 90 degrees and with HRT is to help you carry out your own implementations. On display is a barcode whose HRT additionally contains the check digit calculated by RBarc+ during runtime.

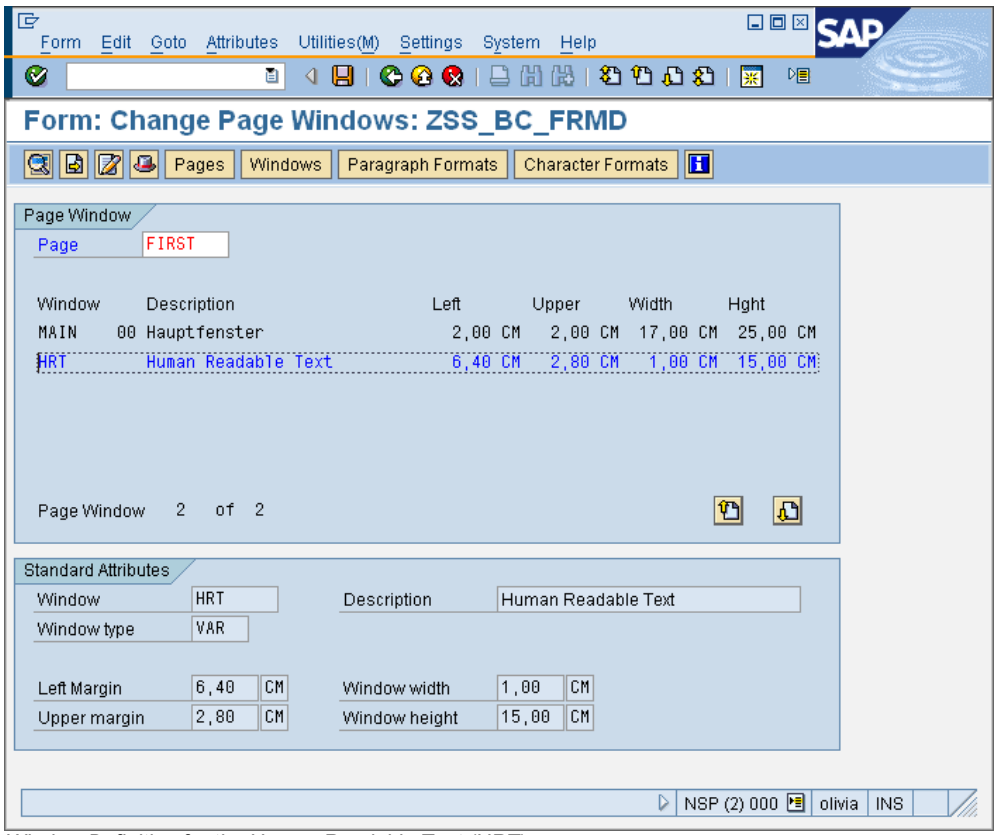
The example form consists of two windows: **MAIN** and **HRT**.

In the window **MAIN**, the barcode was defined, as described in Chapter 5.1.:



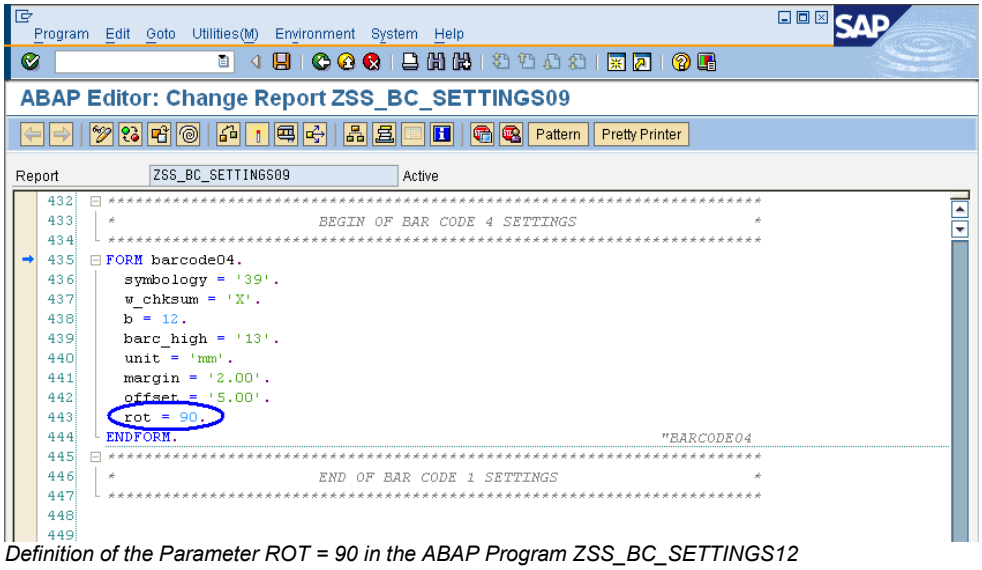
Barcode Implementation in the SAPscript Window MAIN

As we do not wish to output the HRT immediately, the standard return variable **&ENCODING_RETURN&** was buffered in **&HRT1&** (1). That way, it is secured that the value remains unchanged, even if further barcodes are added to the form. The similar is true for variable **&CHECKSUM&,** that contains the current check digit (2). This was buffered in variable **&CHK1&**.



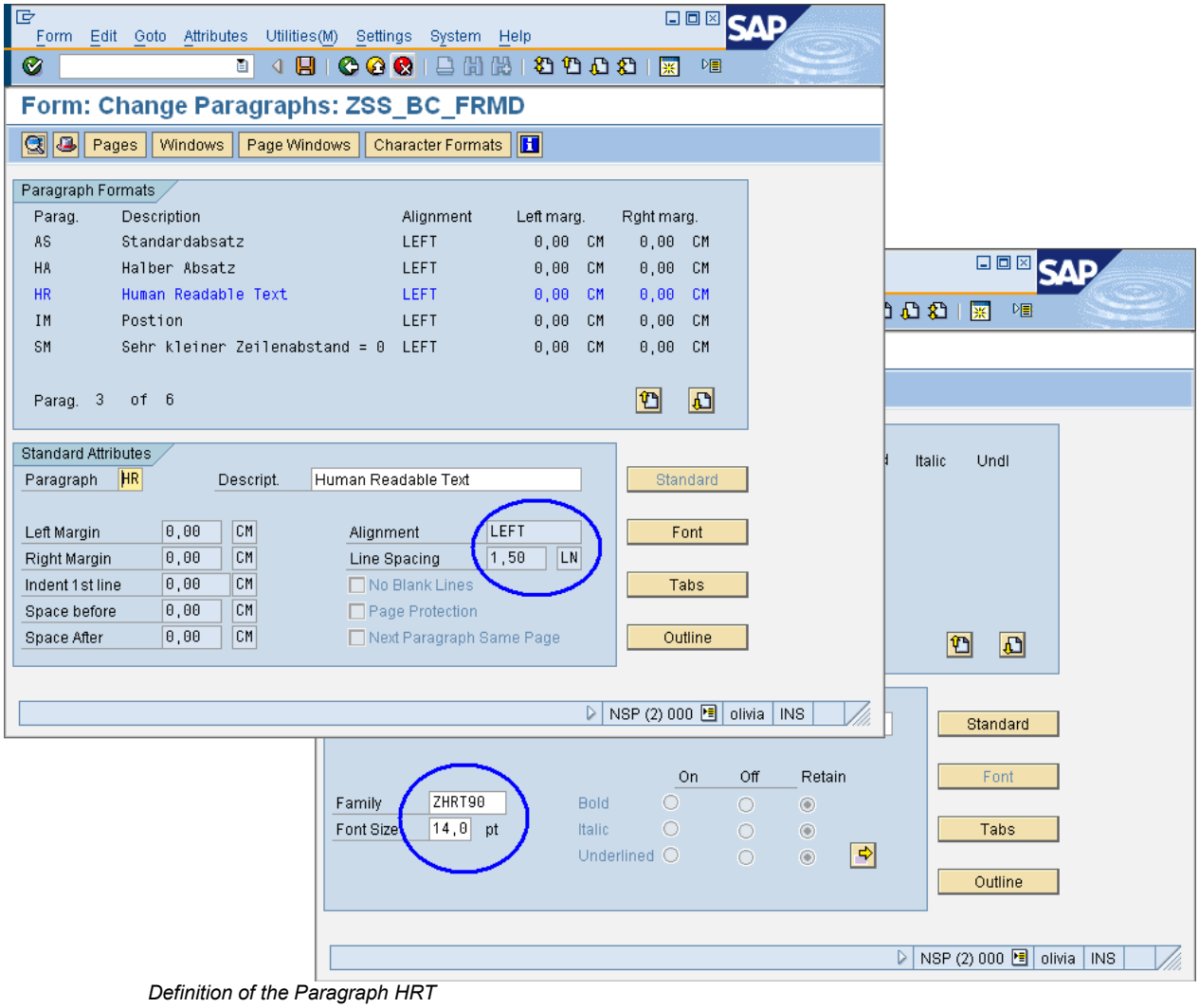
Window Definition for the Human Readable Text (HRT)

To make the barcode rotate by 90 degrees, the parameter **ROT** in program **ZSS_BC_SETTINGS12** was defined accordingly:



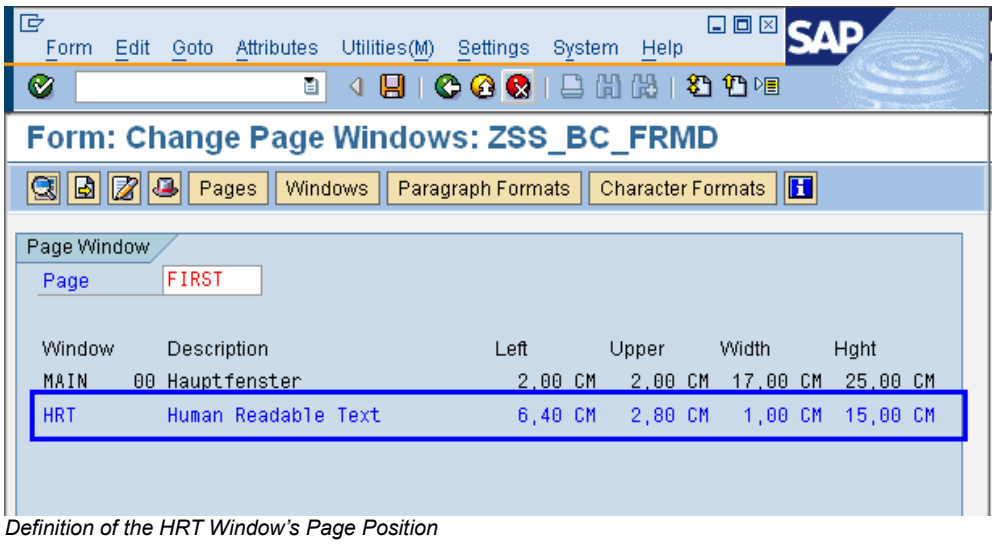
Definition of the Parameter ROT = 90 in the ABAP Program ZSS_BC_SETTINGS12

As the HRT is to be rotated by 90 degrees, paragraph **HR** was defined in the SAPscript form. In the paragraph definition, it was determined that the line spacing for this paragraph is to be **1.5 LN**. Additionally, it was determined that the font type to be used is **ZHRT90** with a height of **14 Pt**.



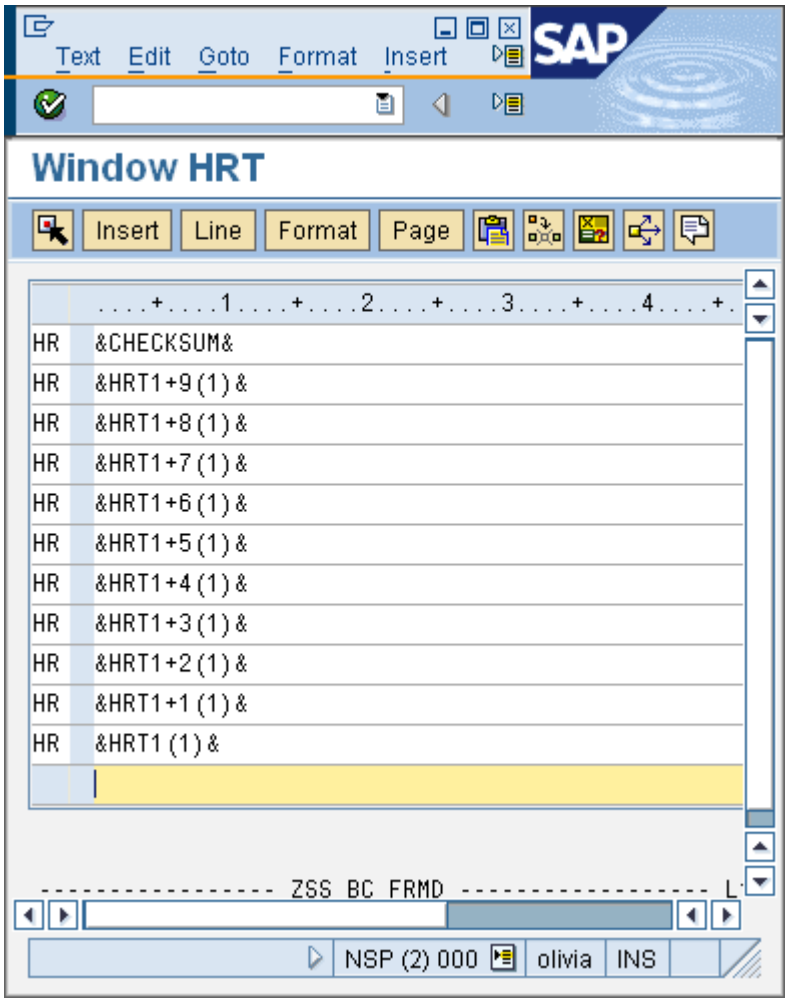
Definition of the Paragraph HRT

The window was positioned in such a way that the HRT will appear directly right to the barcode.



Definition of the HRT Window's Page Position

In the **HRT** window, the **HRT** is output character by character (from back to front) per line (variable **HRT1**). Each line is formatted with paragraph **HR**.



Note: In the SAP pint preview, the result does not appear to be correct in case of rotated HRT. Reason for this is the fact that – for the preview – not the stated True Type fonts but their substitutes are used. However, as soon as you create the print form or a PDF-form, the HRT will be shown it its correctly rotated form.

Result of the SAP-Print Preview

Real Print Result

Note: By using the above procedure, you can also create rotated barcodes with an embedded or semi-embedded HRT. For this, define the parameters **MARGIN** and **OFFSET** in program **ZSS_BC_SETTINGS12** for the corresponding barcode (this will create a white indentation alongside the long barcode margin) and then position the HRT accordingly.

Rotated Barcode with semi-embedded HRT

.

7 Implementation of a Barcode Output in SmartForms

7.1 Embedding Barcodes into a SmartForms Form

Any amount of barcodes can be inserted into one SmartForms form. As here we are dealing with **nodes** of the type **program lines** and **graphics**, the barcode can be inserted anywhere you want: into an existing window, into a separate window or for example into a template or a table. Existing print programs for a SmartForms form will **NOT** be changed. The adjustments only concern the form itself (that way, the RBarc+ solution is SAP-updatable). In order to keep the number of adjustments within the form as low as possible, only the absolutely necessary adjustments are carried out in the SmartForms form – similar to an SAPscript form. Among these is the information about „**what is to be encoded**“? and „**where should the barcode be positioned**“? All other barcode properties, such as **symbology**, **width**, **height,** etc. are conducted in the separate ABAP program **ZSF_BC_SETTINGS12**.

The procedure logics are always the same and conform to the following principle:

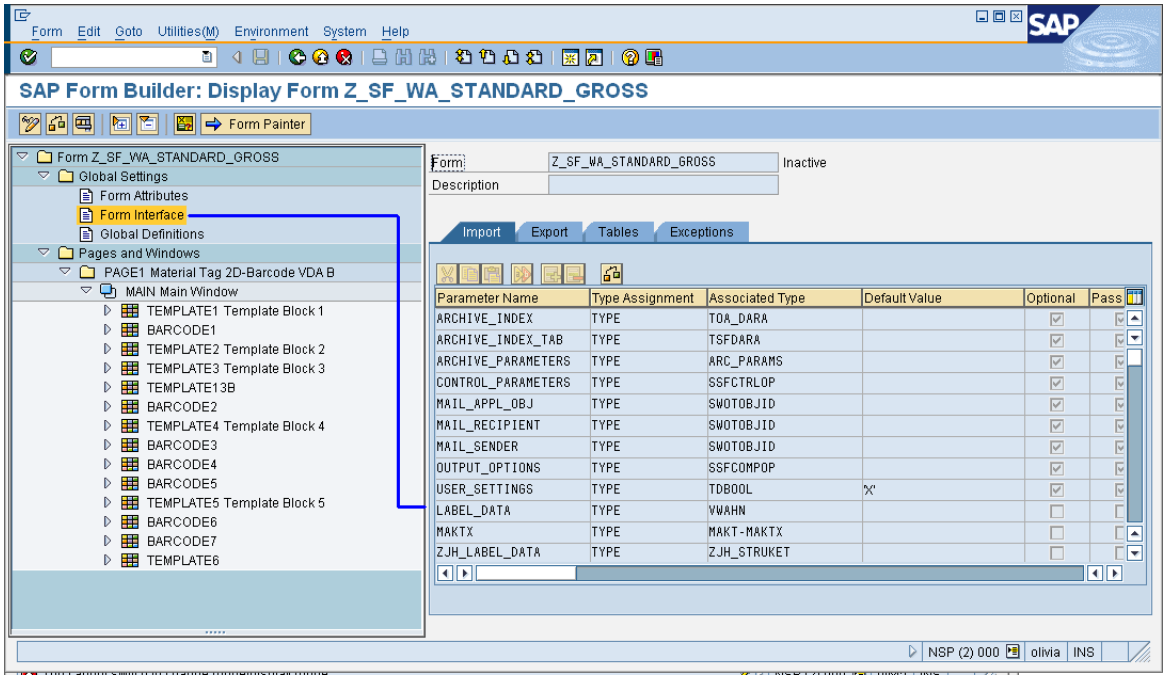
- A one-time definition of the barcode properties (**symbology, height, width,** etc.) in a separate form routine within the ABAP program **ZSF_BC_SETTINGS12**.
- Definition of the variable(s) in the SmartForms form to be encoded as barcodes.
- Determination of a **barcode identification** (each barcode within a form has to be clearly identifiable).
- Calling up the form routine **GEN_BARCODE** in program **ZSF_BC_SETTINGS12** and transfer of the parameters (among which are primarily the data and the barcode identification to be encoded). This will be carried out via a separate program node to be created.
- Taking over the **return parameters** coming back from RBarc+ (primarily the dynamically created **name** of the **barcode graphic** to be embedded).
- Dynamic embedding of the **barcode graphic** into the form via a **graphic node**.
- Deleting the previously created barcode graphic from the system via another programming node.

We will later hear more about the barcode properties, such as **symbology**, **height**, **width,** etc., as that part is identical for both **SAPscript** and **SmartForms** applications.

On the following pages, we will present a screenshot – together with a corresponding explanation – of a SmartForms form with an embedded barcode.

7.1.1 Definition of Variables

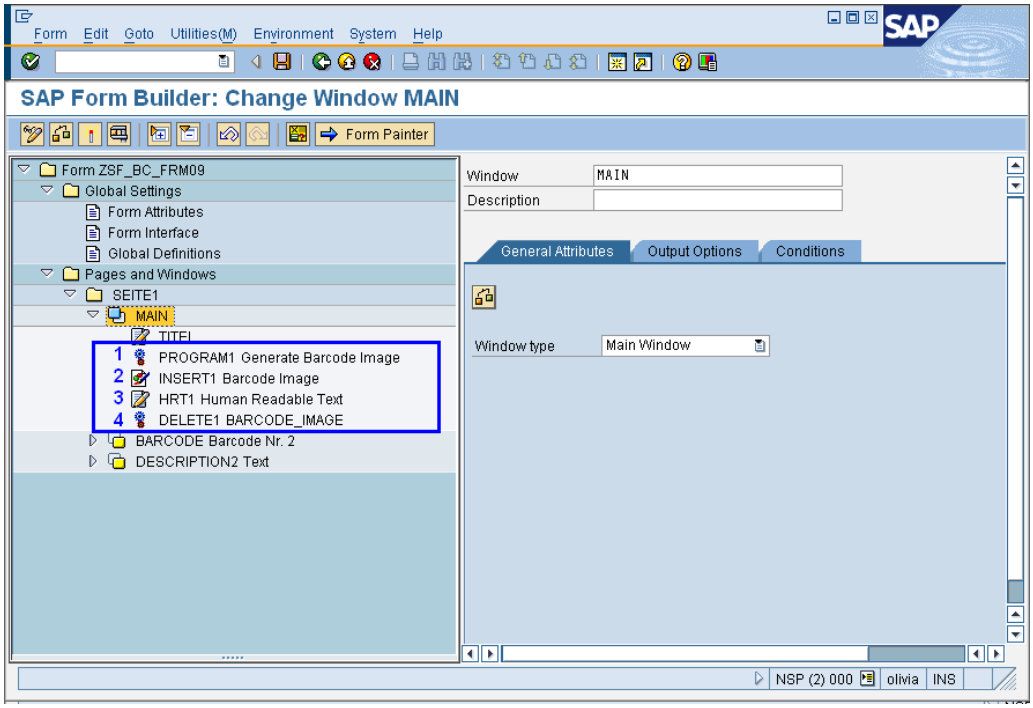
As a rule, changing data originating from processes within a productive environment will get encoded (e.g. delivery notes). These data are received by tables or structures that were defined in the **form interface**. One example for such a definition in the **form interface** could for instance be the structure **LABEL_DATA**, taking reference to the table **VWAHN**. In our example, we will encode the field **VBELN** from the structure **LABEL_DATA** as a barcode.



Example of a Form Interface Definition

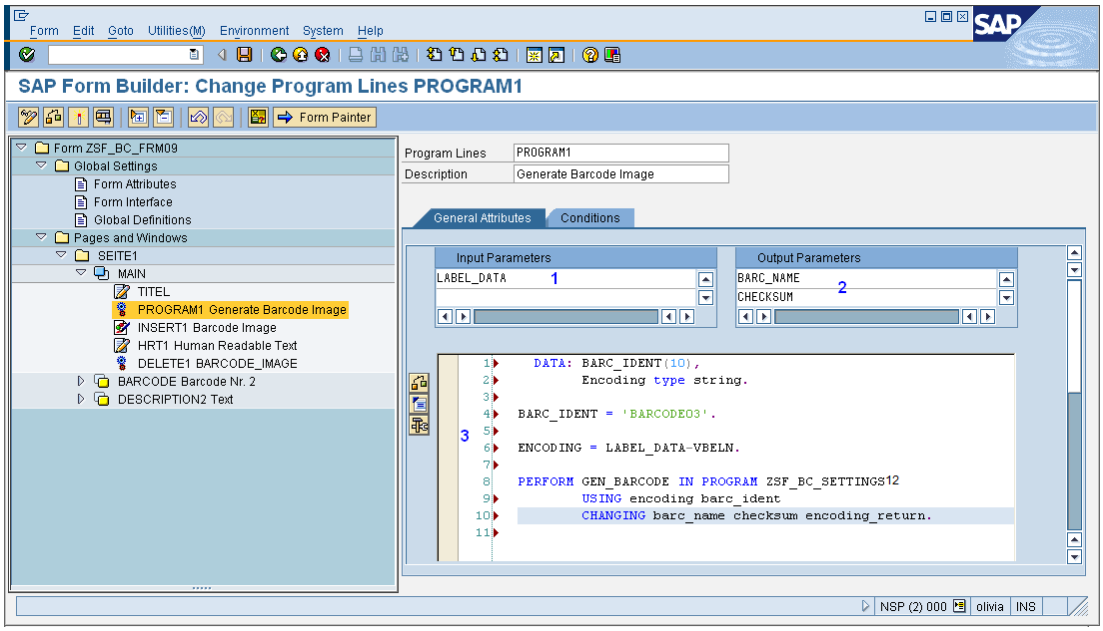
7.1.2 Form Setup for Creating a Barcode

The following image shows you the form structure together with the nodes necessary for the barcode output. In this case, all nodes were inserted into the window **MAIN**. However, as already mentioned, you can use and choose any type of window you like.



Node 1

Program nodes: In this node, the variable is determined which is to be encoded as a barcode and the form routine **GEN_BARCODE** in program **ZSF_BC_SETTINGS12** is called up.



Program Listing of the Barcode Implementation

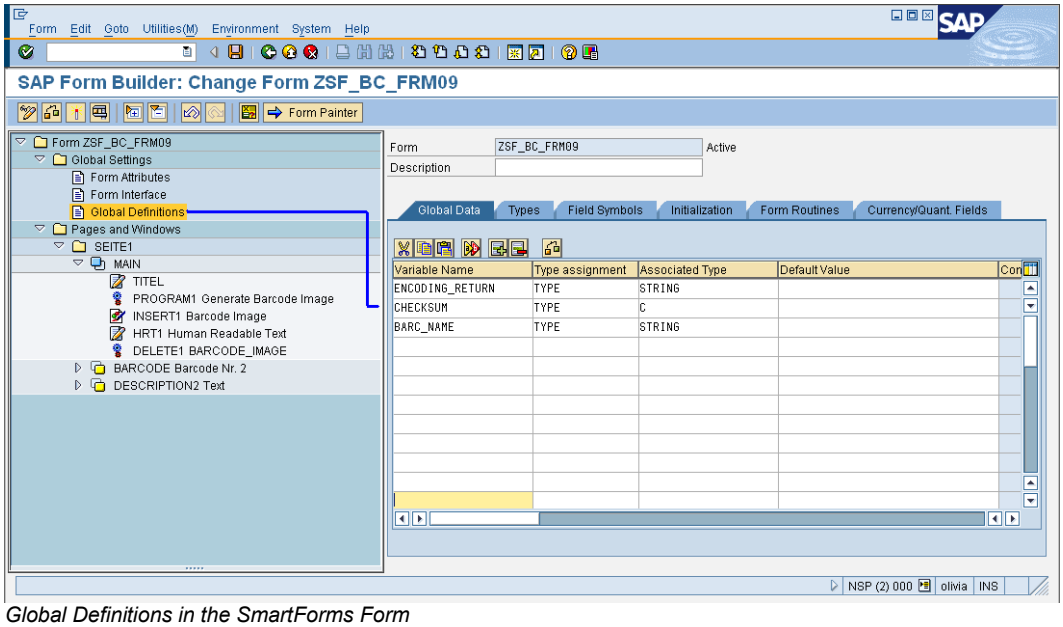
The structure **LABEL_DATA** (1) was defined as the **input parameter**. This is necessary for the variable to be „seen“ from this node.

As output parameters, the variables **BARC_NAME**, **CHECKSUM** and **ENCODING_RETURN** (2) are defined (the latter cannot be seen on this screenshot). This is important, as these variables returned by program **ZSF_BC_SETTINGS12** are needed in the forthcoming form nodes. In order to render these variables viewable all over, they have to be defined in the **global definitions** of the SmartForms form as follows:

BARC_NAME TYPE STRING

CHECKSUM TYPE C

ENCODING_RETURN TYPE STRING



Global Definitions in the SmartForms Form

Program Listing:

Lines 1 and 2:

In the DATA command, the variables **BARC_IDENT(10)** and **ENCODING** are defined.

Line 4

BARC_IDENT = 'BARCODE03'

Here, a barcode identification is allocated that is transferred to RBarc+ by variable **BARC_IDENT**. The maximum length for this variable is 10 characters. Each barcode in the form has to carry a distinct identification, therefore, it is advisable to use a counter as a means of discrimination, as shown in the above example ‚03'.

Note: The barcode identification has to correspond to the name of the form routine in program **ZSF_BC_SETTINGS12**, in which the barcode properties are defined. If necessary, you will have to create your own form routine for a new barcode in program **ZSF_BC_SETTINGS12**. On delivery, 4 form routines are pre-defined in program **ZSF_BC_SETTINGS12**: **BARCODE01, BARCODE02, BARCODE03** and **BARCODE04.** The barcode parameters defined therein can be changed at any time.

Line 6

ENCODING = LABEL_DATA-VBELN

In this line, the variable **ENCODING** is allocated with the value of the field **VBELN** from the structure **LABEL_DATA**. Please keep in mind that, as a rule, barcodes can only encode data of the type **TEXT,** so that figures, especially those containing a dot or a comma, have to be converted to **TEXT** beforehand. You will achieve this by writing the relevant variables of the types Integer or “F” in one text variable.

Lines 8 – 9

```
PERFORM GEN_BARCODE IN PROGRAM ZSF_BC_SETTINGS12
        USING encoding barc_ident
```

Calling up the form routine **GEN_BARCODE** in program **ZSF_BC_SETTINGS12** and transfer of the parameters **ENCODING** and **BARC_IDENT**.

Note: On principle, it is possible to use a separate ABAP program for each form together with the barcode properties defined therein. In this case, please copy program **ZSF_BC_SETTINGS12** into a different program and call this up in the SmartForms form. However, you will have to make sure not to change the name of the variable, as otherwise the program could not function properly.

Line 10

CHANGING barc_name checksum encoding_return.

In **Line 10**, the variables for the return values from Rbarc+ are defined. These names must not be changed. Should you need the values of these variables for a different purpose, it is advisable to write these into separate variables which you have defined yourself.

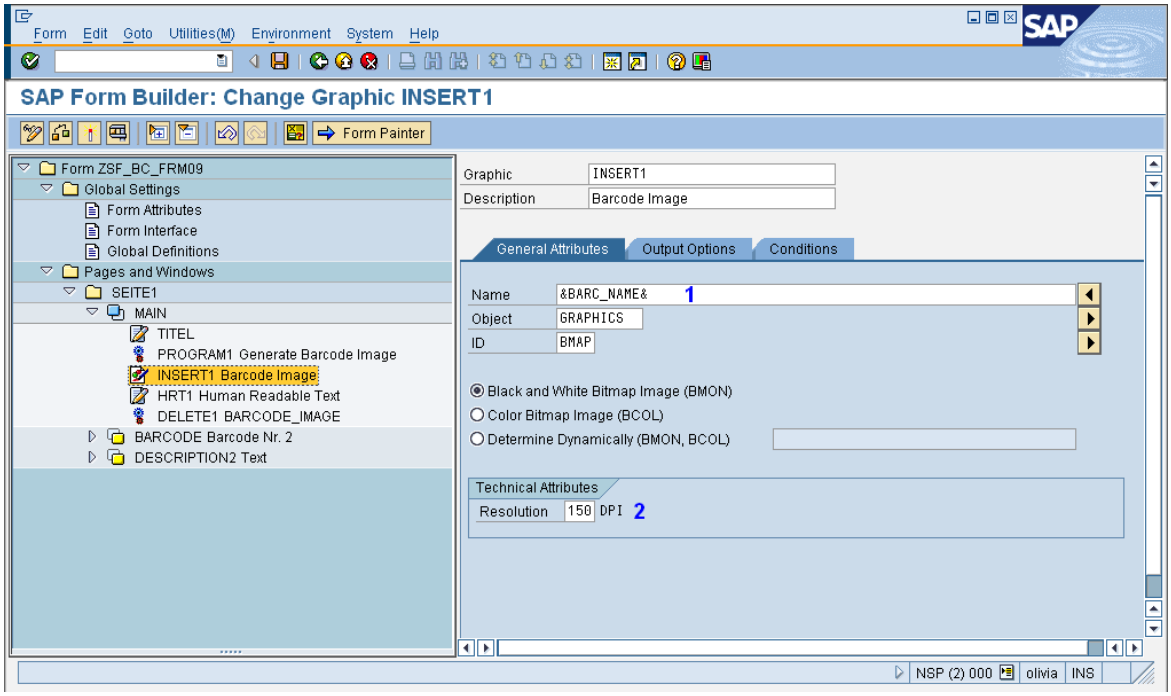
BARC_NAME – name of the graphic created by RBarc+ with barcode image
CHECKSUM – the check digit calculated by Rbarc+ (if required)
ENCODING_RETURN – states what was actually encoded

Note: The returned value for **ENCODING_RETURN** can sometimes deviate from the transferred value for **ENCODING**, for example when certain options, such as the deletion of leading zeros, were set in program **ZSF_BC_SETTINGS12**.

.

Node 2

Graphic Node. In this node, the barcode graphic created by RBarc+ is inserted dynamically. Therefore, in the field *Name* you will find no defined name but the variable **BARC_NAME (1)**. This variable contains the name of the barcode graphic stored in the system created during runtime. This name is disclosed via the **output parameters** of the previous program node. To make everything work properly, the variable **BARC_NAME** has to be stated as a **TYPE STRING** in the **Global Definitions** of the SmartForms form.



The field *Object* has to be filled with the value **GRAPHICS** and the field *ID* has to be filled with the value **BMAP**.

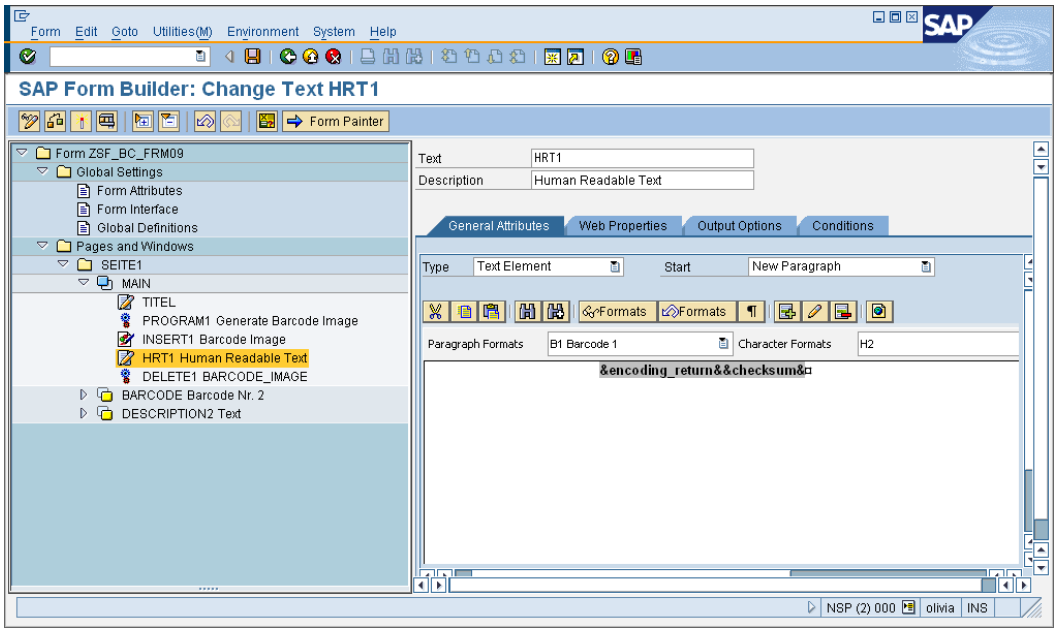
In the field *Resolution,* the actual resolution has to be stated, that was used to create the graphic.

Note: The resolution used to create the barcode graphic is determined in program **ZSF_BC_SETTINGS12** in parameter **RES**. Should the statements in the program and those in the form not be congruent, the result will not correspond to the desired expectations: the graphic shown will either be too large or too small.

Note: For each barcode, a separate resolution can be entered – according to your requirements. With RBarc+, any chosen resolutions can be defined. Should you wish to print documents with barcodes, you should make sure that the output device does really support the graphic resolution you have chosen. If that is not the case, the barcode graphic will be printed in an incorrect size. Well-established laser printers support the following graphic resolutions: 75dpi, 150dpi, 300dpi and 600dpi.

Node 3

Text Node. In this node, the **H**uman **R**eadable **T**ext (**HRT**) is inserted**.**

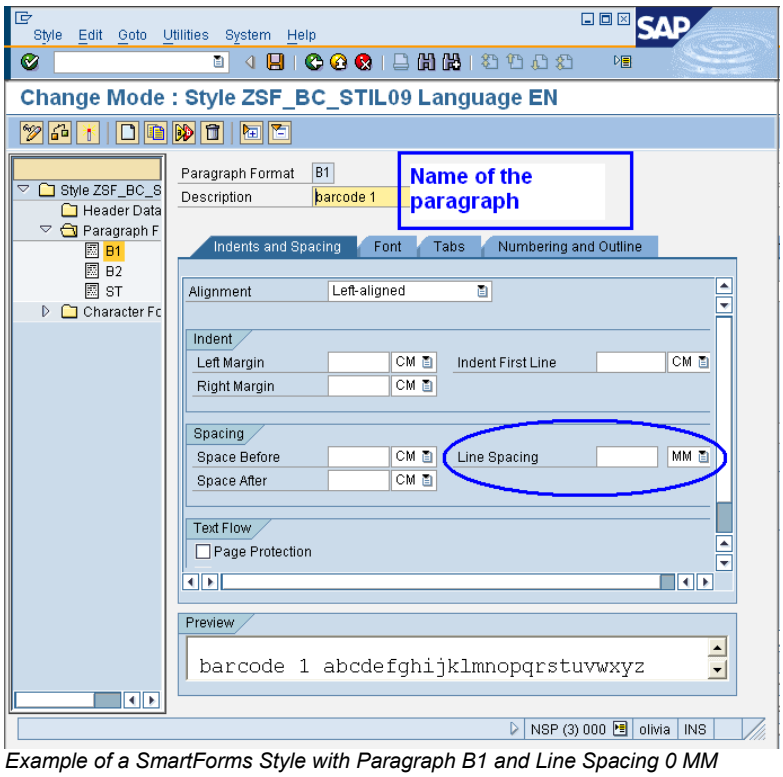


&ENCODING_RETURN&&CHECKSUM&

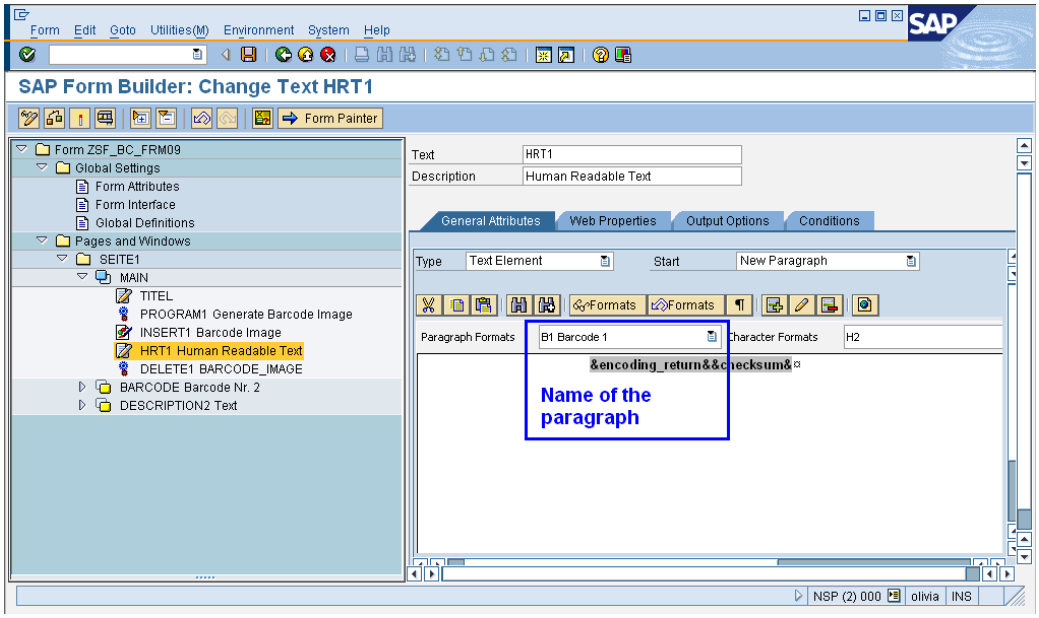
As HRT, the variable **ENCODING_RETURN**, immediately followed by the output of **CHECKSUM**. The horizontal position of the HRT can be achieved by using blank spaces or tab stops.

As a rule, the HRT will appear in the line underneath the barcode graphic:

Sometimes, however, the necessity arises to „embed“ the HRT into the barcode. For this case, you will have to define a margin (**Margin**) for the barcode in program **ZSF_BC_SETTINGS12** and to create a SmartForms style (or expand an existing style). There, you will have to define a paragraph with a line spacing of 0 or for example 0.1 mm – according to your requirements. The style has to be allocated to the text node and the defined paragraph has to be allocated to the HRT:

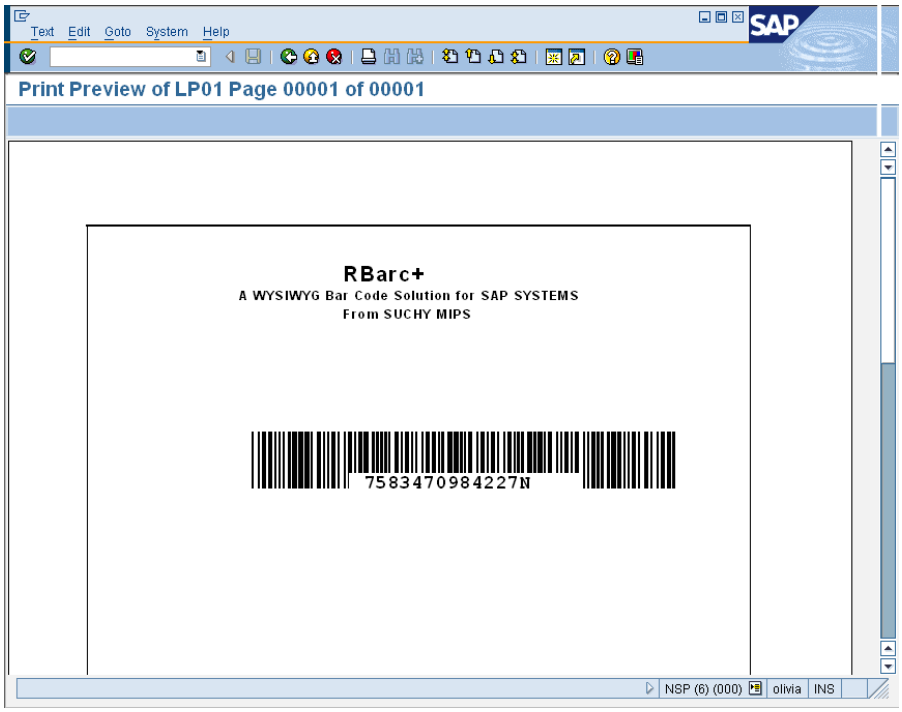


Example of a SmartForms Style with Paragraph B1 and Line Spacing 0 MM



Human Readable Text with allocated Paragraph „B1“

The result should approximately present itself as follows:

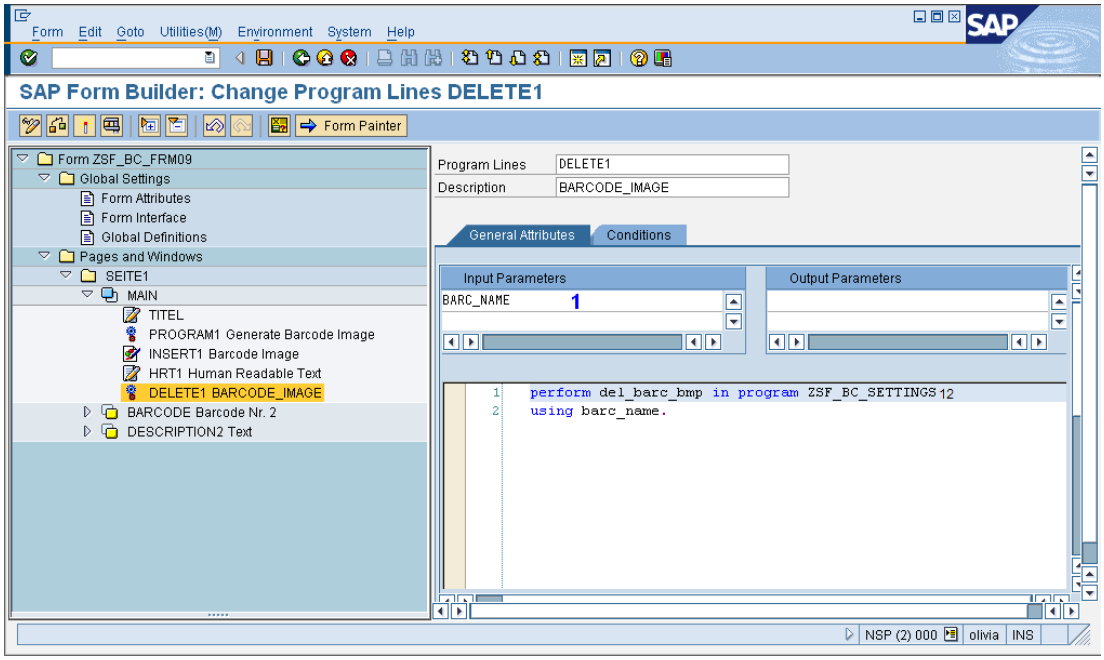


Note: In most cases, the value of the variable **ENCODING_RETURN** corresponds to the variable **ENCODING**. As RBarc+, however, offers additional possibilities to manipulate the input value (**ENCODING**) (e.g., leading blank spaces or zeros can be removed), it is possible that the barcode encodes a different value as stated with **ENCODING**. For this reason, the variable **ENCODING_RETURN** RBarc+ will return the value actually encoded. Example: **ENDODING = 00012345**. As in program **ZSF_BC_SETTINGS12,** the suppression of leading zeros was **set,** only the figures **12345** will actually be encoded. The return parameter **ENCODING_RETURN** will now also receive the value **12345** meant to be used as **HRT**.

Note: On principle, it is possible to output the HRT in a separate window, entirely detached from the barcode. This carries the advantage that no special paragraph formats have to be defined, that the desired position can in fact be reached via positioning the window. Should, in the same form, several different barcodes have to be implemented, you should buffer the original return values in their own variables to make sure that those are not overwritten by the next RBarc+ callup and remain unchanged throughout the entire runtime. For example, for three different barcodes, you define **HRT1**, **HRT2** and **HRT3** and write the value of **ENCODING_RETURN** right into the first program node of each barcode and into the relevant **HRTx** variable. This way, each HRT will remain unchanged for the entire form operation and can be employed at any time.

Node 4

Program Node. In this node, the previously created barcode is deleted. Of course, the barcode must only be deleted from the system after it had already been inserted into the form. Therefore, please make sure that this node comes after the graphic node within the program flow logic.



Please define **BARC_NAME** as your **Input Parameter (1)**.

Program Listing

Lines 1 - 2

perform del_barc_bmp in program ZSF_BC_SETTINGS12
using barc_name.

This command calls up the form routine **del_barc_bmp** from program **ZSF_BC_SETTINGS12**, which deletes the previously created barcode with the name **BARC_NAME** from the system.

Note: Should you wish to output several barcodes in one form and should you have defined the HRT in separate windows, you will certainly have buffered the value of the HRT in its own variable (e.g. **HRT1**). In this case of course, you will have to enter that variable as an input parameter (1) instead of **BARC_NAME**, and as a USING-Parameter in program line 2.

7.2 Definition of the Barcode Properties

As already mentioned in previous chapters, the barcode properties are defined in a separate ABAP program. These definitions are identical for both **SAPscript** and **SmartForms**. An SAPscript, however, requires a different interface for any data transfer than SmartForms does, for SAPscript the program **ZSS_BC_SETTINGS12** is used; for SmartForms, the program is **ZSF_BC_SETTINGS12**. Please make sure that you are really using the right program. In the following, we will only refer to those form routines that are equal for both programs.

7.3 Insertion of a Rotated Human Readable Text (HRT) within a SmartForms Form

Similar to SAPscript, SmartForms does not offer the possibility to output rotated dynamic text. For those of you who are only interested in SmartForms and have skipped Chapter 5.3, we will repeat once again the information contained in Chapter 5.3. Should you, however, have already read Chapter 5.3., you can now skip Chapter 6.2 and carry on with the ensuing chapters.

7.3.1 TrueType Fonts with Rotated Characters

In order to enable printing of rotated HRT, we have developed 3 special font types, with the characters already rotated by **90**, **180** or **270** degrees. Should you have correctly followed all the installation steps, these fonts should already be installed on your SAP system. Should that not be the case, please return to the installation instructions and install the TrueType fonts **ZHRT90**, **ZHRT180** and **ZHRT270** that were part of the delivery.

Example HRT90:
1 2 3 4 5

Example HRT180
12345

Example HRT270
1 2 3 4 5

As the character feed is still carried out by the system in horizontal direction despite the font rotation, for a vertically oriented text, each character from the HRT has to be output in a separate line:

1
2
3
4
5

As you will have certainly already noticed, the sequence of the figures is inverted compared with the original character sequence 12345. Therefore, in case of text output that was rotated by **90** or **180** degrees, you will have to output the HRT characters in the reversed sequence. For this, SAPscript offers the appropriate possibilities described in the following:
Let us assume that the HRT has a length of 10 characters as was stored in variable **HRT1.** Then, in case of a text rotated 90 or 180 Grad degrees – seen from above or from the left – first the tenth, then the ninth, then the eighth, etc. character has to be printed. You can achieve this with the following encoding:

(Example HRT1 = '0123456789')

Command Lines for 90 degrees	**Result with a 90-degree rotation**
&HRT1+9(1)&	9
&HRT1+8(1)&	8
&HRT1+7(1)&	7
&HRT1+6(1)&	6
&HRT1+5(1)&	6
&HRT1+4(1)&	5
&HRT1+3(1)&	4
&HRT1+2(1)&	3
&HRT1+1(1)&	2
&HRT1(1)&	1

Command Line

&HRT1+9(1)&&HRT1+8(1)&&HRT1+7(1)&&HRT1+6(1)&&HRT1+5(1)&&HRT1+4(1)&&HRT1+3(1)&&HRT1+2(1)&HRT1+1(1)&&HRT1(1)&

Result with a 180-degree rotation

9876543210

In case of a 270-degree rotation, the command sequence has to be in reverse of the 90-degree rotation.

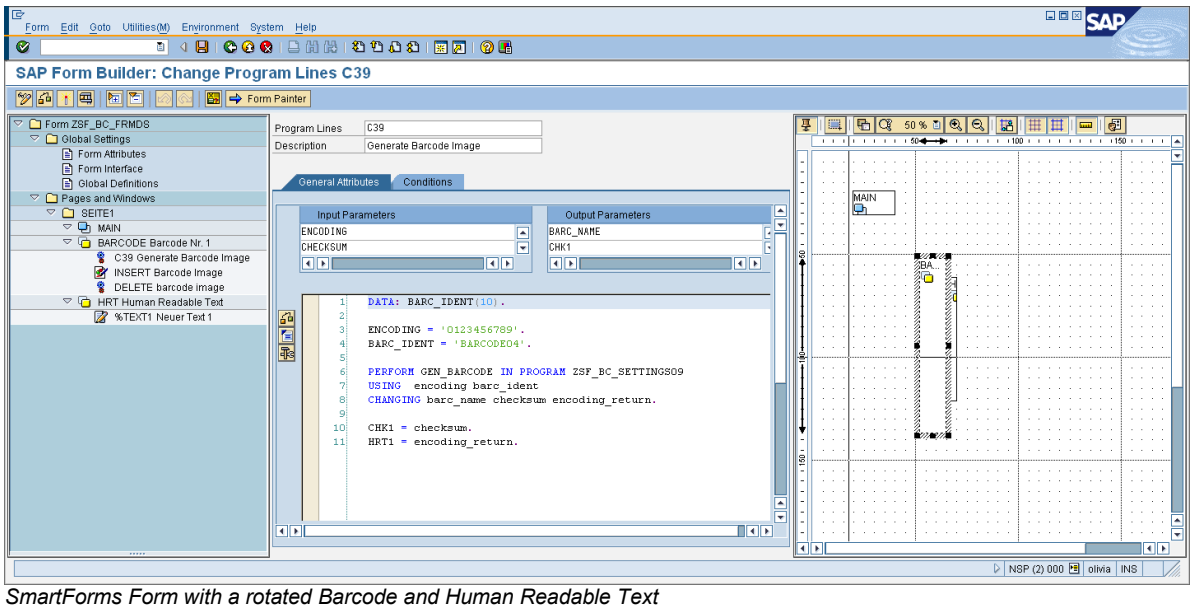
Command Lines for 270 degrees	**Result with a 270-degree rotation**
&HRT1+(1)&	0
&HRT1+1(1)&	1
&HRT1+2(1)&	2
&HRT1+3(1)&	3
&HRT1+4(1)&	4
&HRT1+5(1)&	5
&HRT1+6(1)&	6
&HRT1+7(1)&	7
&HRT1+8(1)&	8
&HRT1+9(1)&	9

Note: Should it, in case of a 90 or 270-degree rotation, be necessary to set the single characters closer or further apart, you will have to define a relevant paragraph in the suitable style and use this to format all paragraphs of the HRT characters.

As it is often difficult to pack all this encoding into an existing SmartForms window without interfering with the layout, we suggest using a separate window for the HRT. This has the additional advantage that positioning the HRT can easily be carried out via the window position.

The following example of a SmartForms form with a barcode rotated by 90 degrees and with HRT is meant to help you carry out your own implementations. A barcode is on display whose HRT additionally contains the check digit calculated by RBarc+ during runtime.

The example-form consists of three windows: **MAIN, BARCODE** and **HRT**, with **MAIN** carrying no significance for this example.

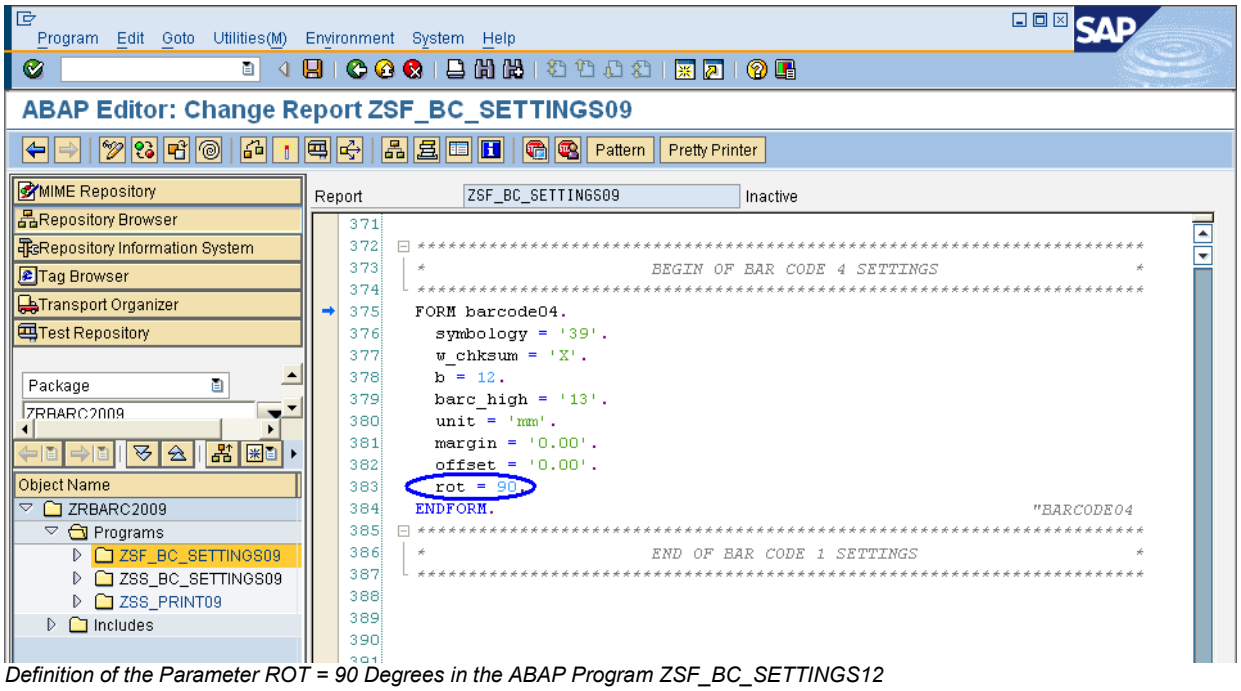


SmartForms Form with a rotated Barcode and Human Readable Text

In the window **BARCODE**, the barcode was defined as described in Chapter 6.1.
The window **HRT** was positioned in such a way that the HRT appears directly to the right of the barcode.

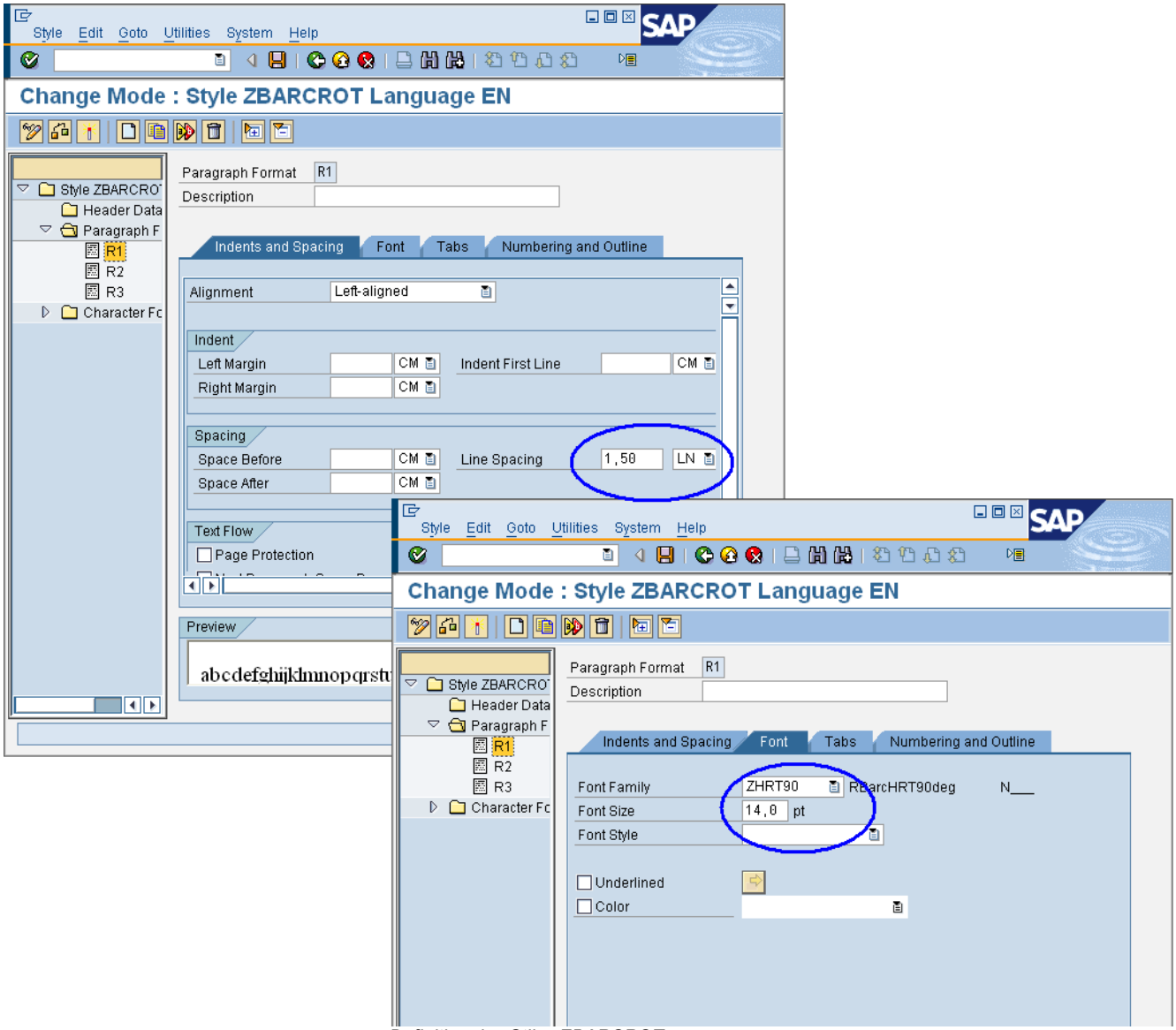
As we do not wish to output the HRT immediately, the standard return variable **ENCODING_RETURN** was buffered in variable **HRT1** (1). Thus, it is ascertained that the value remains unchanged, even if further barcodes should be created in the same form. The similar is true for variable **CHECKSUM** containing the current check digit (2) which is buffered in variable **CHK1**.

For the barcode to rotate by 90 degrees, parameter **ROT** in program **ZSF_BC_SETTINGS12** was defined accordingly:



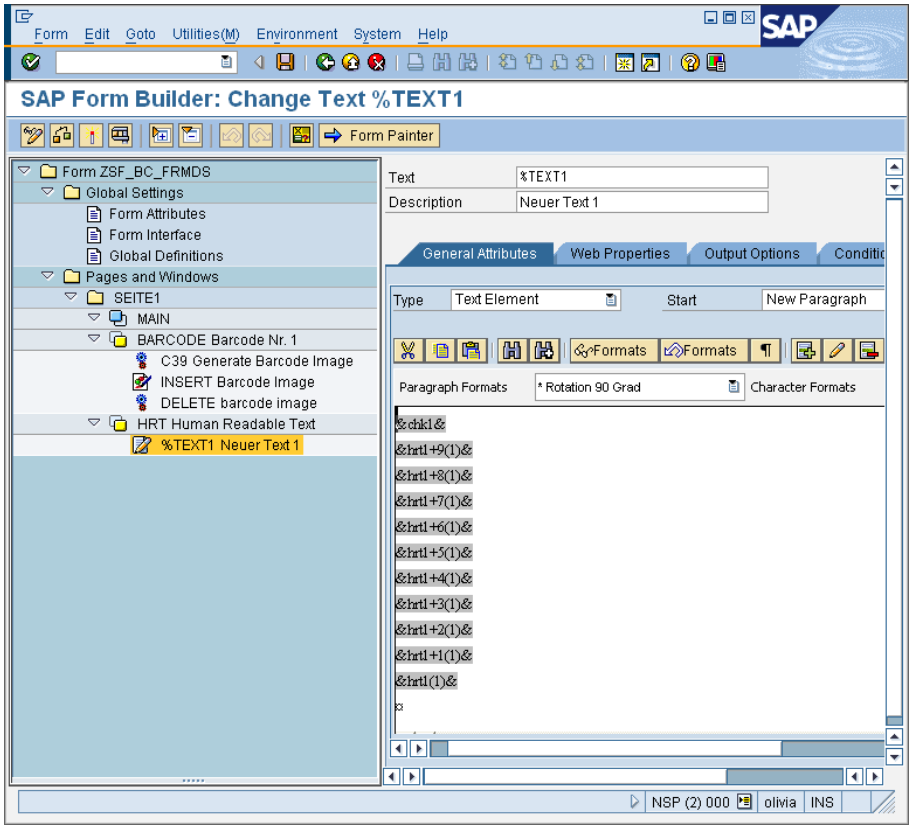
Definition of the Parameter ROT = 90 Degrees in the ABAP Program ZSF_BC_SETTINGS12

As the HRT is to be rotated by 90 degrees, a style with the name **ZBARCROT** was created. There, the paragraphs **R1** – for 90 degree, **R2** for 180 degree and **R3** for 270 degree rotations were created. The paragraphs were allocated to the relevant TrueType fonts **ZHRT90**, **ZHRT180** or **ZHRT270** with a size of 14 **Pt**. and a line spacing of **1.5 LN**.

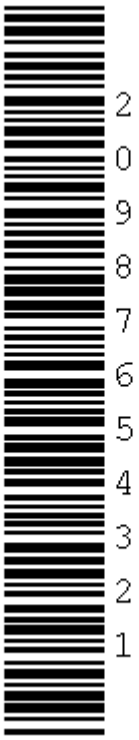


Definition des Stiles ZBARCROT

In the window **HRT**, the text node is allocated to the style **ZBARCROT**. In the text node, the HRT (variable **HRT1**) is output character after character (from back to front) per line. Each line was formatted with paragraph **R1**.



Note: The result in case of rotated HRT in the SAP print preview appears to be incorrect. Reason for this is the fact that for the print preview not the stated True Type fonts but their substitutes are used. As soon as you print the form or create for example a PDF-document, the HRT will be on display in its correctly rotated form.



Result of the SAP-Print Preview

Real Printing Result

Note: Using the above-described procedure, you can also design rotated barcodes with embedded or semi-embedded HRT. For this, please define the parameters **MARGIN** and **OFFSET** in program **ZSF_BC_SETTINGS12** for the relevant barcode (that will create a white indentation alongside the long barcode margin) and position the HRT accordingly.

Rotated Barcode with Semi-Embedded Human Readable Text

.

8 Implementing a Barcode Output with AdobeForms

8.1 Using RBARC with AdobeForms

In the standard Adobe Life Cycle Designer SAP supplies the usual barcodes. In which situations does it still make sense to use RBarc+?

- **The desired barcode type is not available in AdobeForms.**

In its object library, Adobe Forms does not offer all barcodes. Barcodes, that are not available in AdobeForms, can be created with RBarc+.

- **AdobeForms does not print the barcode in the desired quality.**

Especially when printing barcodes at dynamic positions of the main window (e.g. output of position data) the barcode created by Adobe Forms frays on top and bottom. In this case, RBARC is able to print barcodes with a dynamic positioning in a much better quality.

- **With AdobeForms the barcodes cannot be parameterized exactly.**

Sometimes the business partners specify exact parameters for printing a barcode. With Adobe Forms the height can be exactly specified and it is also possible to draw the barcode narrower and wider, however, an exact specification of the module width is not possible. With RBARC you can use the entire palette of parameters in order to create the barcode precisely according to exact definitions.

- **AdobeForms does not return the checksum.**

Adobe Forms does create the barcode including the checksum but does not return the checksum to the calling program. With RBarc+ you can print the barcode with the checksum and you can evaluate the calculated checksum in the calling program.

8.2 Functionality of creating Barcodes in AdobeForms with RBarc+

At first the ABAP printing program or an itemized ABAP-Coding calls the interface to the RBarc+ in the interface at the time of its initialization and transfers the value the barcode should carry together with the barcode type. In this case, the barcode type includes the parameters for the barcode output, e.g. barcode type (Code39, …), barcode height, module width, calculation of the checksum, etc.

RBarc+ now temporarily creates the barcode as a bitmap in the SAP graphic data base (transaction SE78) and returns it as a binary object of the type XSTRING to the calling ABAP- program. Following this, the temporary bitmap is deleted from the SAP graphic data base.

8.3 Embedding Barcodes into an AdobeForms Form

Any amount of barcodes can be embedded into an Adobe Forms form. The adjustments are concerning the interface and the form only, so that existing printing programs do not have to be changed. Thus, the RBarc+ solution is compatible for SAP updates.

To keep the form adjustments as low as possible, only the utmost necessary definitions are made. Especially necessary is the information: „**what is to be coded**“? All other barcode properties such as **symbology**, **width**, **height** etc. are carried out in the external ABAP program **ZAF_BC_SETTINGS**.

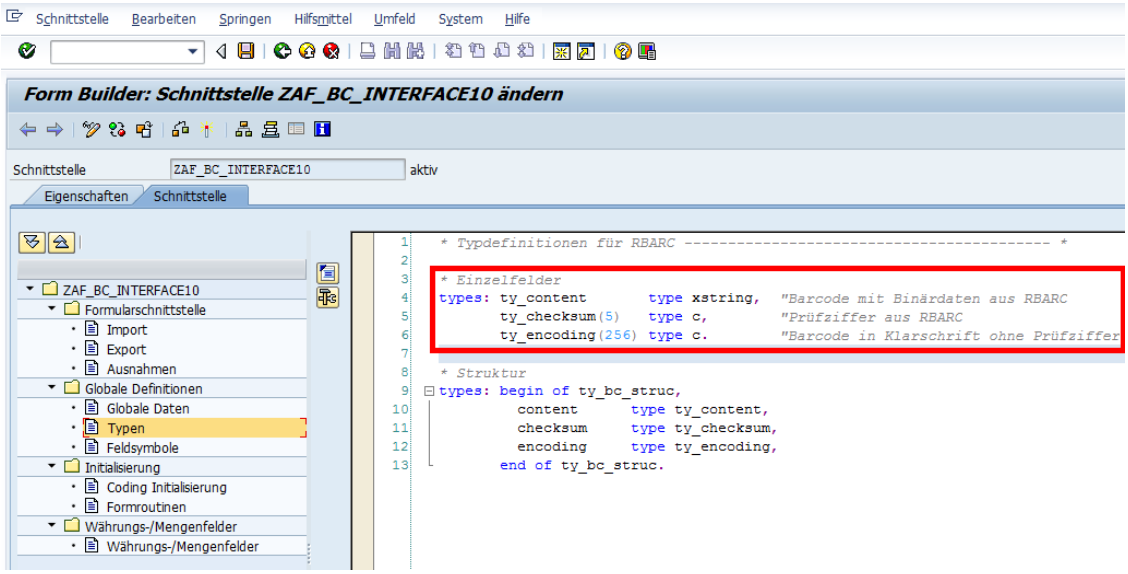
About the properties of the barcodes such as **symbology**, **width**, **height** etc**.** we will write further down in **Chapter 9 „Generating Barcodes“**, as that part is identical for **SAPscript**, **SmartForms** and **AdobeForms** applications. In the following, we present a listing of an **AdobeForms** interface and an **AdobeForms** form together with the relevant explanations that will certainly enable you to embed barcodes into **AdobeForms** forms with RBarc+.

The logic of the procedure is always the same and is based on the following principle:

- **Creating global data types in the interface**

At first, the following 3 data types have to be created in the knot „Global Definition→ Types“:

```
types: ty_content        type xstring,   "Barcode mit Binärdaten aus RBARC
       ty_checksum(5)    type c,         "Prüfziffer aus RBARC
       ty_encoding(256)  type c.         "Barcode in Klarschrift ohne Prüfziffer
```

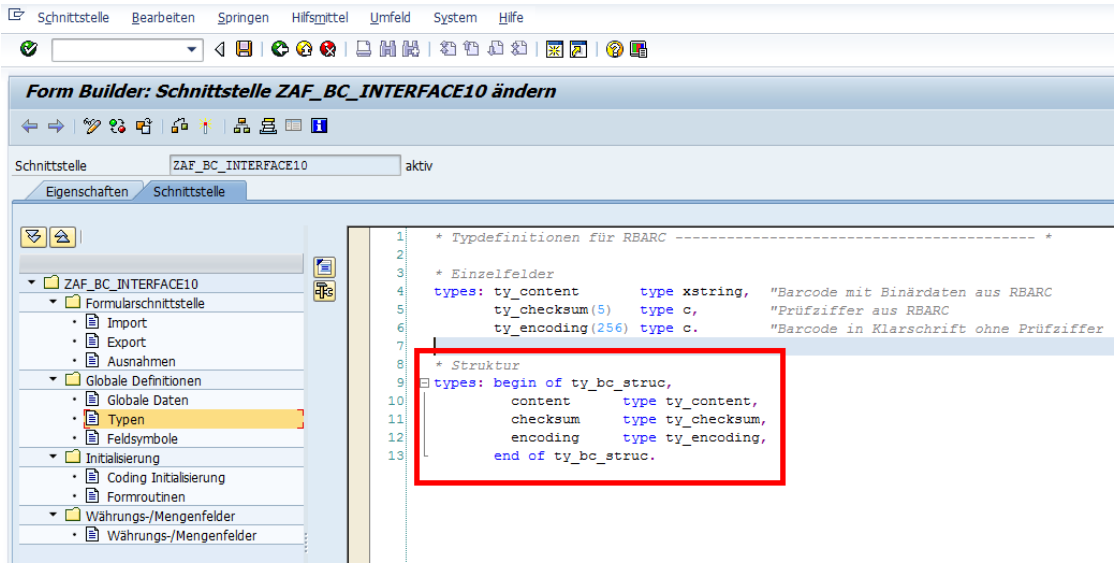


Then, you define the global type definition for a structure.

The structure includes the following 3 fields:

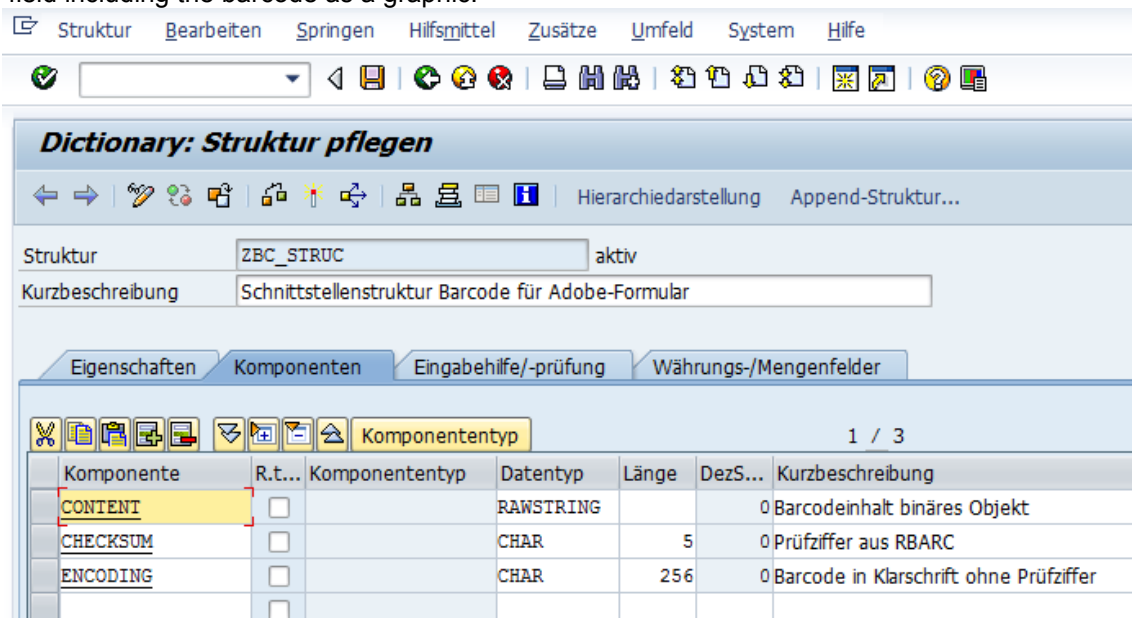
a) One field including the barcode created by RBarc+ as a graphic with binary data,
b) One field including the value to be encoded in the barcode,
c) One field including the checksum calculated in RBARC.

```
types:  begin of ty_bc_struc,
                content      type ty_content,
                checksum     type ty_checksum,
                encoding     type ty_encoding,
        end of ty_bc_struc.
```



Analogously, expanding the structure can also be carried out in the Data Dictionary, when in the interface a structure from the Data Dictionary is used, that is completed in the super ordinate printing

program. In this case the type RAWSTRING (corresponds to the elementary ABAP data types) is used for the field including the barcode as a graphic.



You can cal also insert the single fields into existing structures of an interface.

- **Creating a global structure in an Interface**

  Now, you will create a global structure in the interface in knot „Global Definition→ Global Data“:

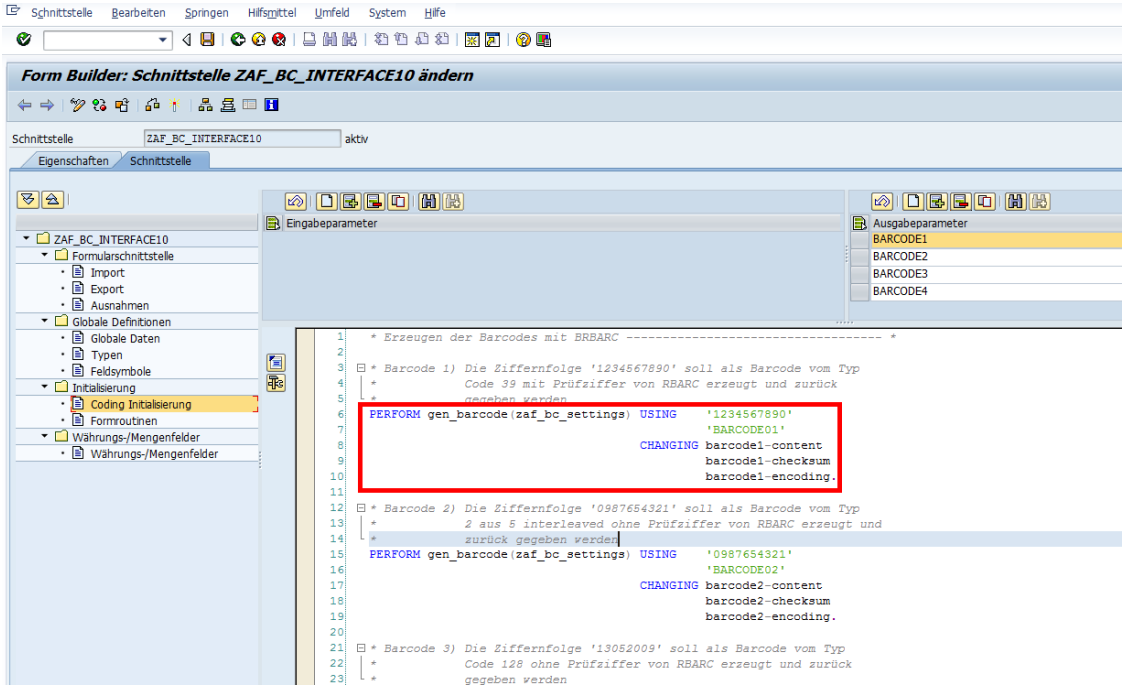
  The global structure is of the same type of the just created structure definition.

  In the example, we define a global structure BARCODE1 of the data type TY_BC_STRUC. The global structure includes the 3 above-stated variables barcode contents, checksum and barcode in human readable form.

- **Calling RBarc+ in the Initialization of the Interface**

  In the knot „Initialization → Coding Initialization“ of the interface, now the interface to the RBarc+ is called in order to create the barcode.

```
PERFORM gen_barcode(zaf_bc_settings) USING    '1234567890'
                                              'BARCODE01'
                                     CHANGING barcode1-content
                                              barcode1-checksum
                                              barcode1-encoding.
```



The interface shows 2 USING-Parameters and 3 CHANGING-Parameters that have the following meanings:

USING-Parameter 1: The value to be encoded in the barcode.
Here, you always enter the value of the barcode to be encoded. This is for example an order number, material number, and batch or lot number.
In the example, a fixed value is given.

USING-Parameter 2: Barcode type with all properties and parameters created in RBarc+.
USING-Parameter 2 includes a routine that is called in the program ZAF_BC_SETTINGS (in this example) and includes all parameters for the barcode, as for example the barcode type (Code39, …), the barcode height, die module width, etc. This routine (in the example: BARCODE01) has to be created in the called program including the interface to RBarc+ (in the example program ZAF_BC_SETTINGS).

CHANGING-Param. 1: Barcode contents = binary object, including the barcode as a graphic. The graphic always includes only the barcode, but not the value of the barcode in human readable form.

CHANGING-Param. 2: Checksum calculated in RBarc+.

CHANGING-Param. 3: The value encoded in the barcode in human readable form without checksum.
As the barcode contents always include the graphic but never the value of the barcode in human readable form, it is advisable to save the value encoded in the barcode in a separate field.

- **Expanding the Program ZAF_BC_SETTINGS in the SE38**

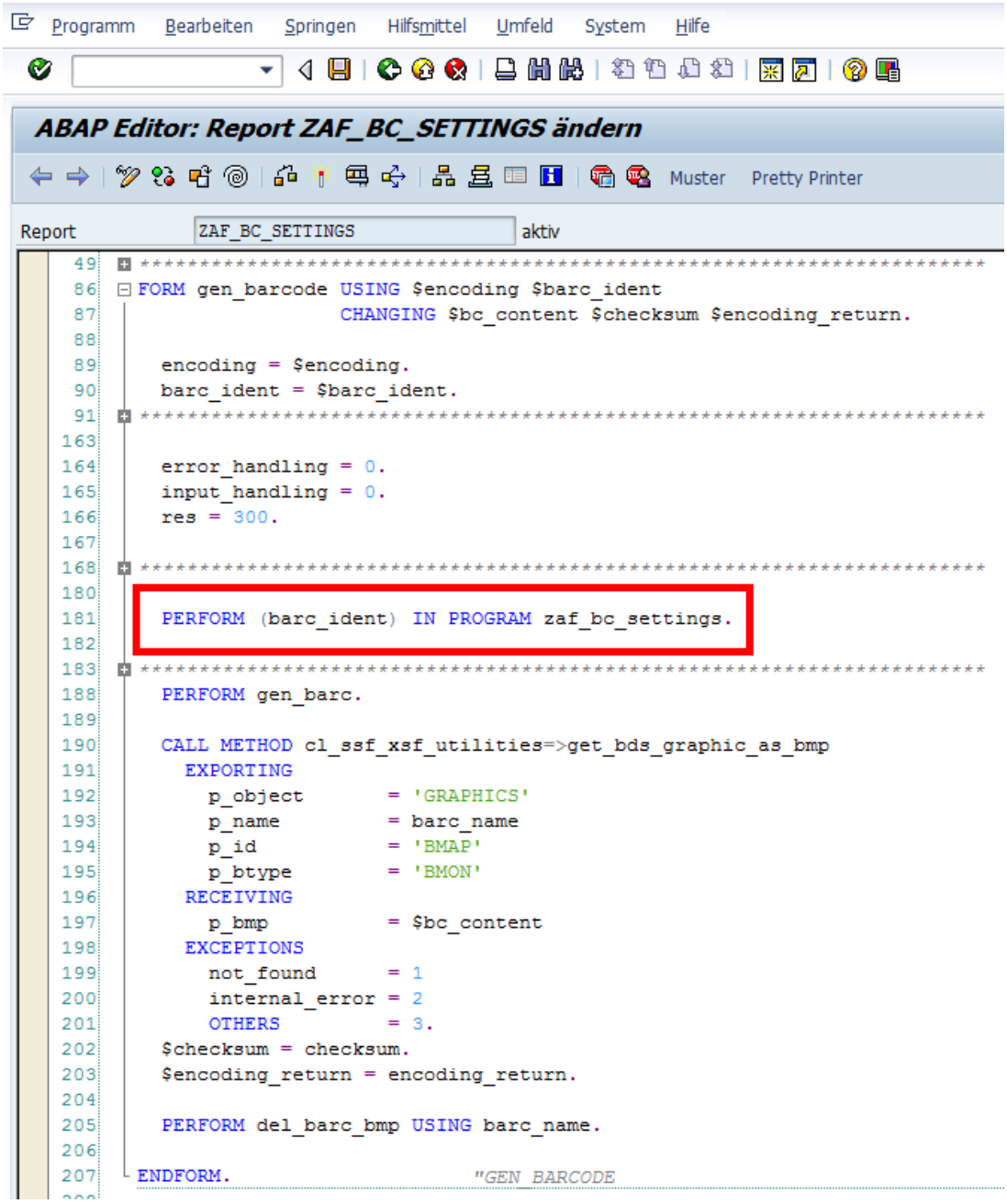
In the program ZAF_BC_SETTINGS the properties of the used barcodes (such as barcode type, barcode height, module width, etc.) are defined. Additionally, it includes the central interface to the RBarc+, where the barcode is eventually created as a graphic.

The program ZAF_BC_SETTINGS is included in the delivery and should now already be installed.

What now has to be done in program ZAF_BC_SETTINGS is again explained according to the example program ZAF_BC_FORM. Final goal is to print a row of figures in the form head as a barcode with the following properties:

Barcode type: Code 39 with checksum
Barcode height: 13mm

In the Adobe Forms interface the routine GEN_BARCODE of program ZAF_BC_SETTINGS is called.




```
49  ************************************************************************
86  FORM gen_barcode USING $encoding $barc_ident
87                   CHANGING $bc_content $checksum $encoding_return.
88
89    encoding = $encoding.
90    barc_ident = $barc_ident.
91  ************************************************************************
163
164   error_handling = 0.
165   input_handling = 0.
166   res = 300.
167
168 ************************************************************************
180
181   PERFORM (barc_ident) IN PROGRAM zaf_bc_settings.
182
183 ************************************************************************
188   PERFORM gen_barc.
189
190   CALL METHOD cl_ssf_xsf_utilities=>get_bds_graphic_as_bmp
191     EXPORTING
192       p_object       = 'GRAPHICS'
193       p_name         = barc_name
194       p_id           = 'BMAP'
195       p_btype        = 'BMON'
196     RECEIVING
197       p_bmp          = $bc_content
198     EXCEPTIONS
199       not_found      = 1
200       internal_error = 2
201       OTHERS         = 3.
202   $checksum = checksum.
203   $encoding_return = encoding_return.
204
205   PERFORM del_barc_bmp USING barc_name.
206
207 ENDFORM.                    "GEN_BARCODE
```

The variable ENCODING includes the value to be encoded in the barcode, the variable BARC_IDENT includes a form routine, which must be called in order to create the barcode with the parameters filed in the variables. Associated to the field BARC_IDENT is a sub program contained in program ZAF_BC_SETTINGS that you will have to create and that you have to supply with values.

In our example, the field BARC_IDENT includes the value BARCODE01. Through the ABAP command

```
PERFORM (barc_ident) IN PROGRAM zaf_bc_settings.
```

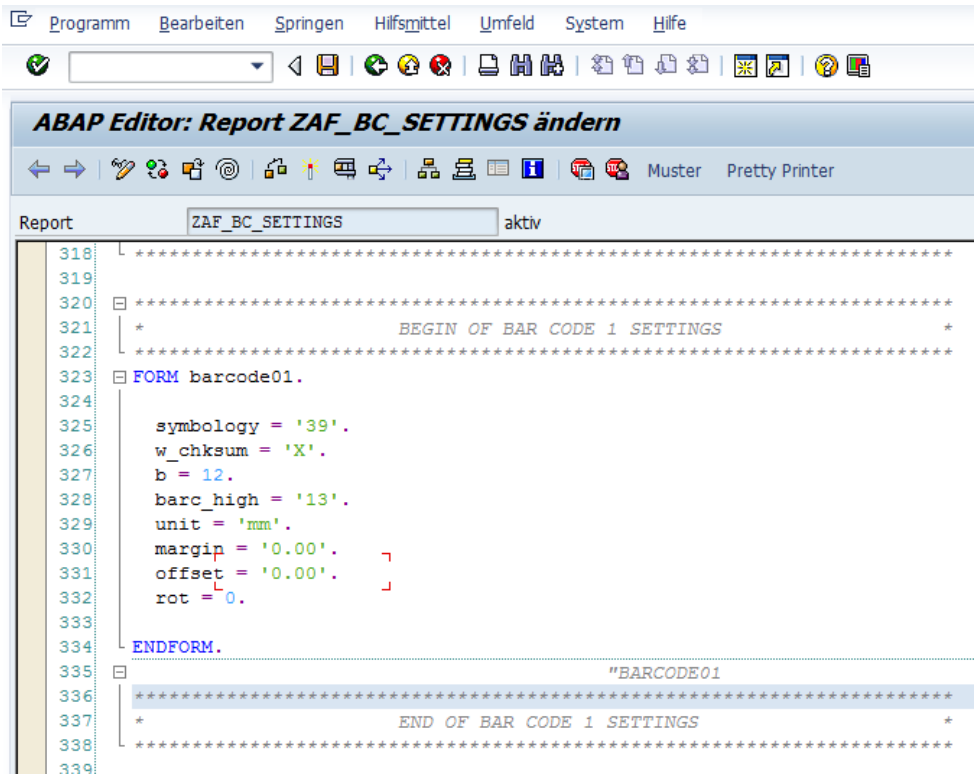
the routine BARCODE01 is called, which can be found further down in program ZAF_BC_SETTINGS.

```
FORM barcode01.

  symbology = '39'.
```

```
  w_chksum = 'X'.
  b = 12.
  barc_high = '13'.
  unit = 'mm'.
  margin = '0.00'.
  offset = '0.00'.
  rot = 0.

ENDFORM.
```

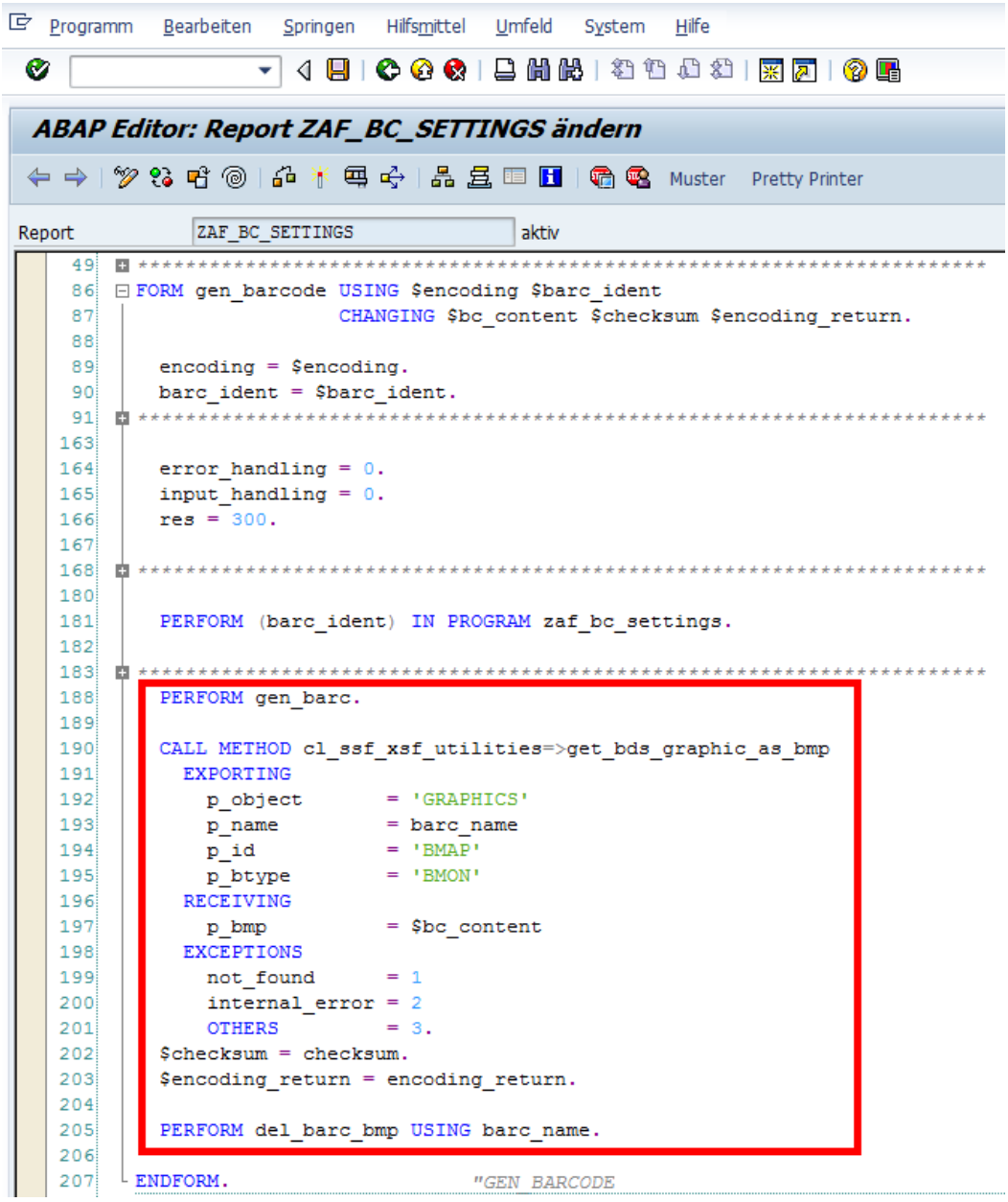


The following parameters are set:

symbology = ‚39‘	→ Barcode of type Code128-A is created
w_checksum = ‘X’	→ Barcode is created with checksum
barc_high = ‚13‘	→ Barcode height = 13mm

All further settable parameters, please find in the RBarc+ documentation.

With routine GEN_BARC the barcode is created in RBARC.



The method

```
CALL METHOD cl_ssf_xsf_utilities=>get_bds_graphic_as_bmp
```

draws the created barcode as a binary object from the graphic data base and completes the field BC_CONTENT, which then transfers the barcode to the calling program.

In routine

```
PERFORM del_barc_bmp USING barc_name.
```

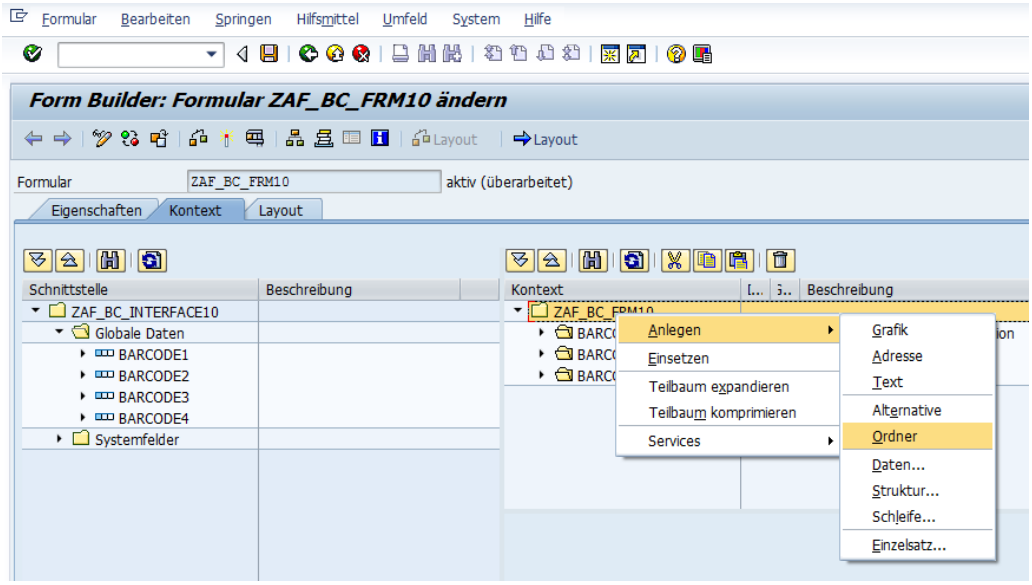
the created barcode will then be deleted.

- **Expansion of the Form Context**

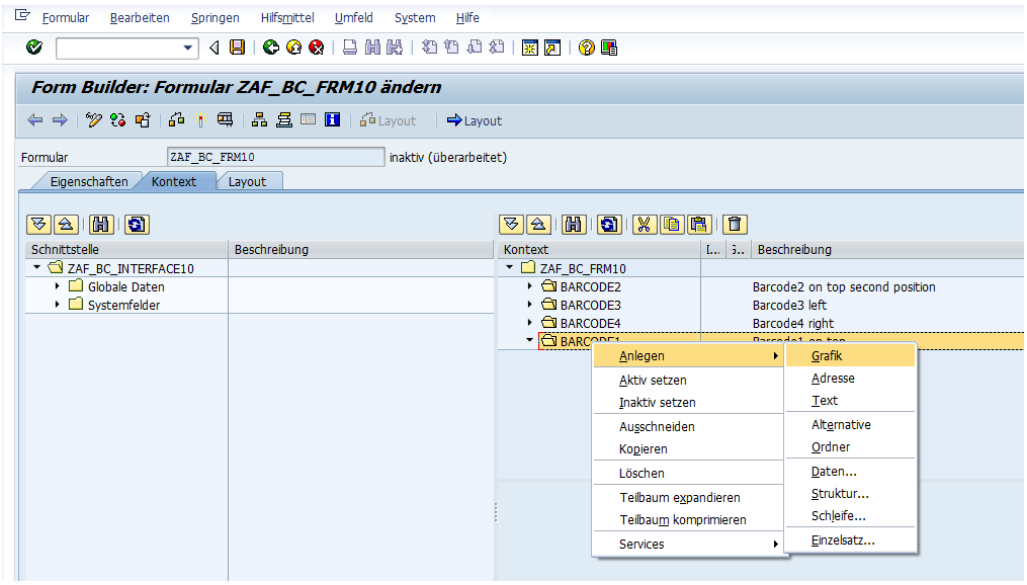
  In the Adobe Form Builder the barcodes created with RBarc+ are shown as an image object.

  Therefore, the context has to be expanded by the elements necessary for a barcode.

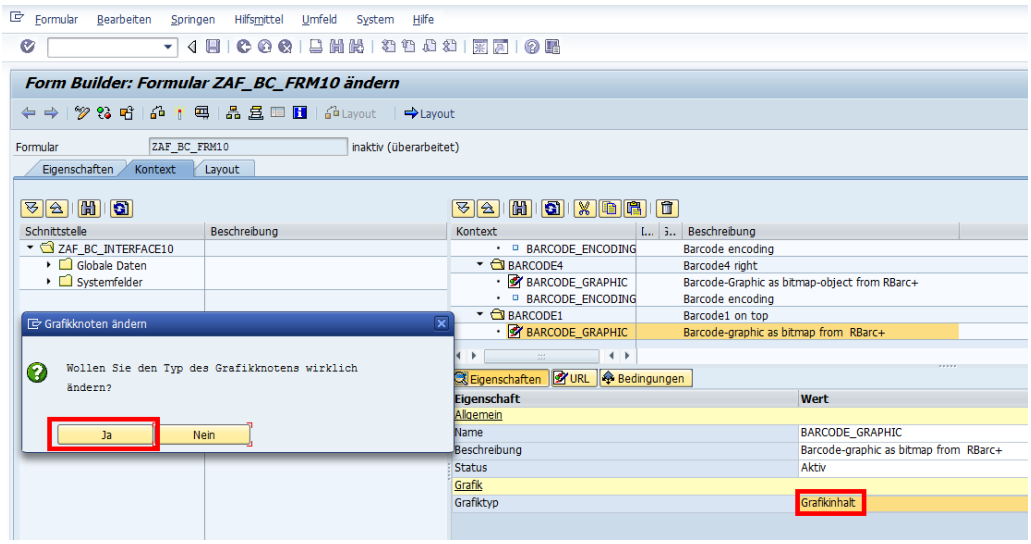
  In the form (transaction SFP) you will now move into the context and create a file.
  Enter a file name and an explication. In the example, the file is called BARCODE1.



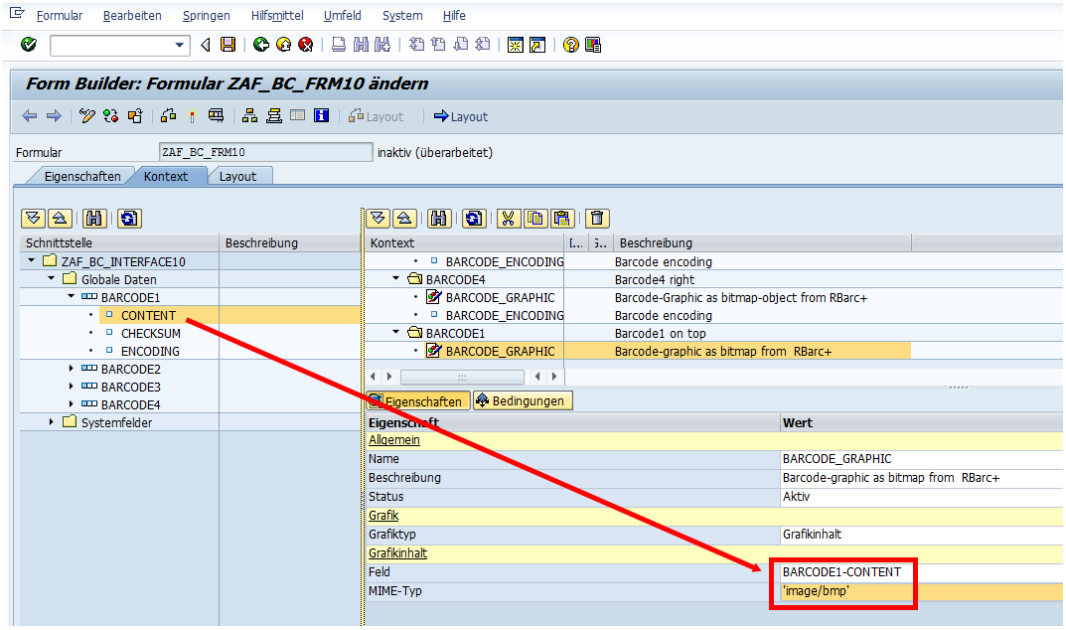
Now, you will create a graphic in file BARCODE1.
Enter a graphic name and an explication. In the example, the graphic is called BARCOD_GRAPHIC.



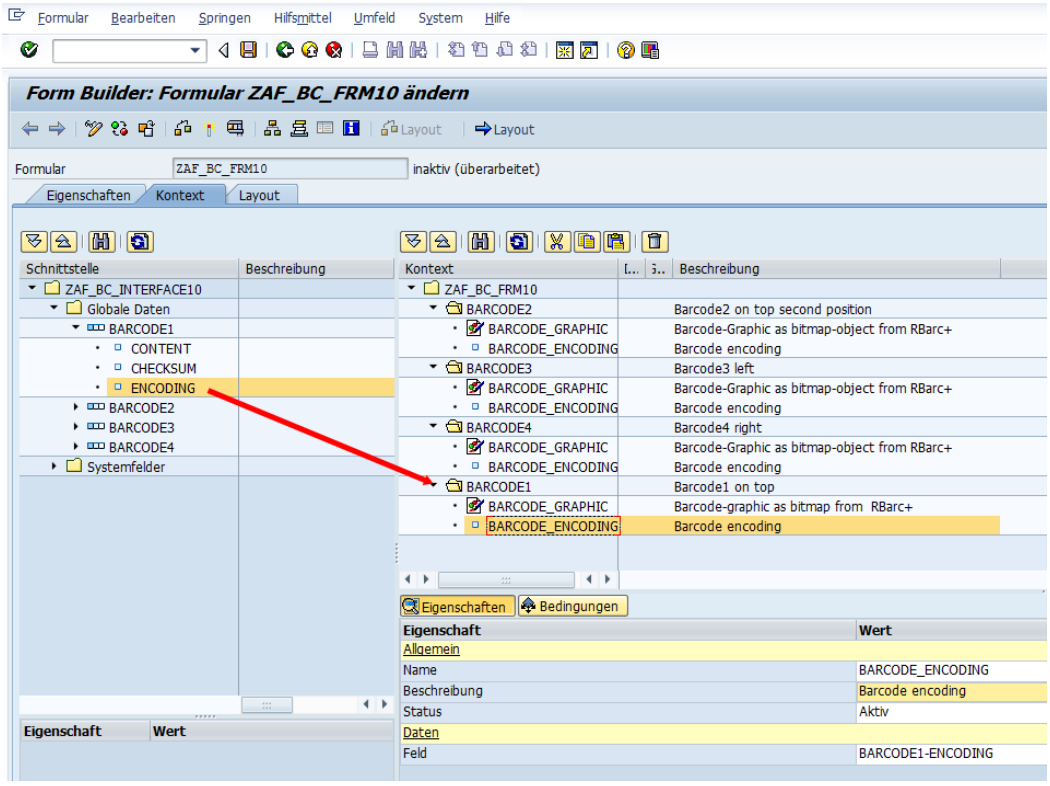
Now, change the graphic type to „graphic content“ and confirm the popup with „Yes“.



Now, allocate the field to the graphic that contains the barcode content as a binary data stream (in the example: BARCODE1-CONTENT) and provide it with ‚image/bmp' as an MIME-Type.

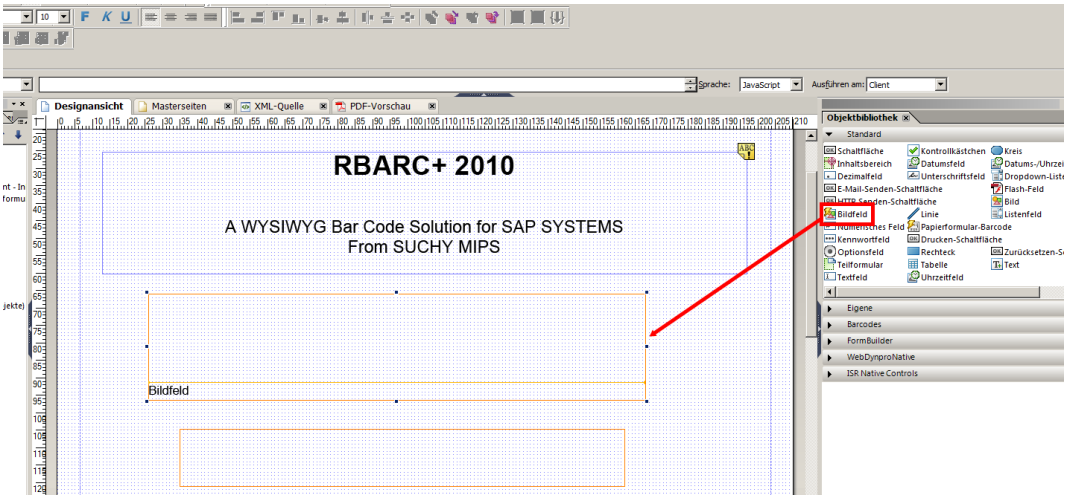


Subsequently, create a data field in the file BARCODE1 by pulling, with the left mouse button pressed, the field BARCODE1-ENCODING from the interface over to the file BARCODE1. This field then includes the barcode in human readable form.

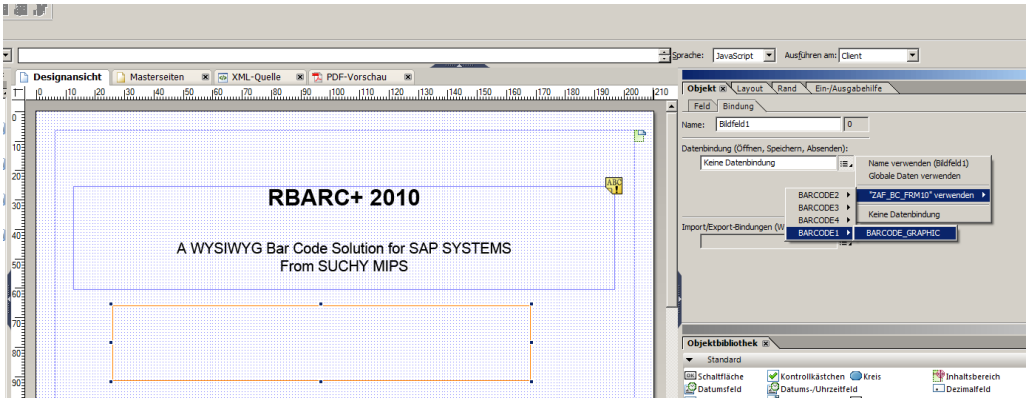


- **Expanding the layout in the Form**
  Now, change over to the layout view for the Form Builders.

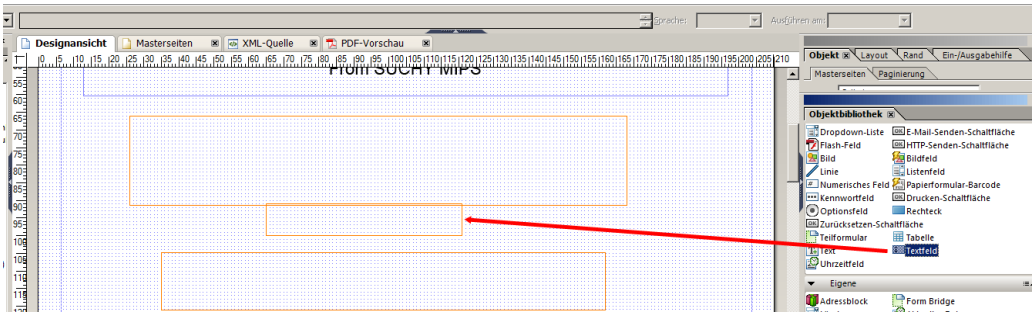
  Create an element of the type „image field".



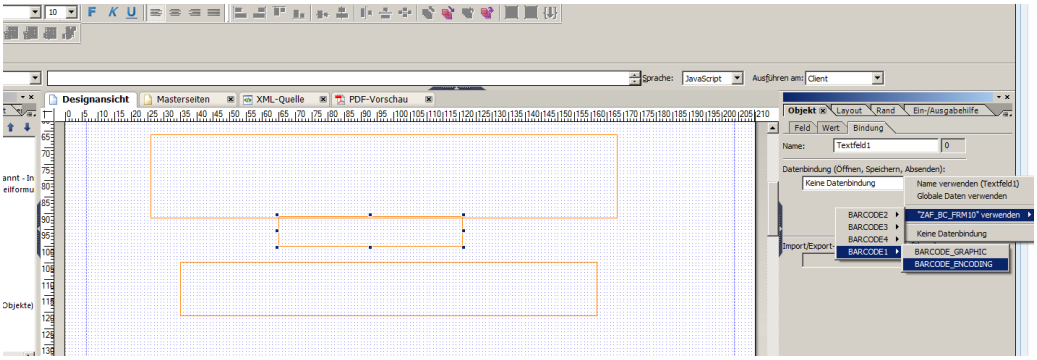
As a data connection, provide the image field with the graphic knot defined in the context (in the example: BARCODE1.BARCODE_GRAPHIC).



Now, create a field of the type „text field“ in human readable form for the value of the barcode below the just created image field.



As a data connection, provide the text field with the field of that structure, containing the barcode value in human readable form (in the example: BARCODE1.BARCODE_ENCODING).



Activate the form – you can now test it.

9 The Barcode Generation

With each implementation of a barcode output, either with SAPscript, SmartForms or with AdobeForms, the form routine **GEN_BAROCDE** is called up. For SAPscript, this form routine can be fond in program **ZSS_BC_SETTINGS12** and in **ZSF_BC_SETTINGS12** for SmartForms. Both programs are identical apart from the interfaces to the forms, which are laid out differently with SAPscript than with SmartForms. For this reason, we will refer to the **SETTINGS** program, meaning the appropriate program relevant in each case.

The **SETTINGS** program is responsible for generating the barcode requested by the form and for returning its name to the form.

The general program procedure is always the same and proceeds according to the following scheme:

Step 1

After having started the form routine **GEN_BARCODE** from the form name, global settings are carried out that determine the general characteristics of RBarc+, e.g. when errors occur.

Step 2

In the second step, the variable **BARC_IDENT** transferred from the form is evaluated and branched out to the form routine that carries the same name. When you implement new barcodes, you will have to make sure that the value for **BARC_IDENT** (e.g. **BARCODE06**) stated in the form is identical with the name of the form routine in which the basic barcode characteristics were determined.

Step 3

As a third step, the barcode is generated by the RBarc+ routines and entered into the system. What will be returned are the barcode name, the check digit (if the barcode was created with a check digit) and the actually encoded value.

Step 4

At the end, the return parameters **BARC_NAME** (name of the barcode), **CHECKSUM** (check digit) and **ENCODING_RETURN** (the actually encoded value) are returned to the activating form

9.1 Definition of Global Parameters for the Flow Control

Among the global parameters (as a standard valid for all barcodes to be created) are the 4 following parameters:

Graph_type Determines the type of graphic. Standard for SAPscript is **<0OTF>** and for SmartForm it is **<0BMP>**. As a rule, this should not be altered.

Error_handling Determines how errors are to be handled by RBarc+.

The permitted values are:

0 In case of an error, the program stops and displays a message window with the relevant error message. After confirmation of the message by the user, the program recovers the error in case it is uncritical (for erroneous parameters such as e.g. the height, standard values are set) so that the barcode can be created and the program continues. Should a critical error have occurred, an error message is generated, the program will continue but no barcode is created.

1 Should a non-critical error occur, it is automatically recovered and there will be no error message. In case of critical errors, an error message will be displayed in a message window. After the user has confirmed the message, the program continues but no barcode is created.

2 No error messages are shown. In case of non-critical errors, these are automatically recovered. In case of critical errors, a black square is on display instead of the barcode.

The list of all critical and non-critical errors can be found on the following page:

Critical Errors

Errors that cannot be recovered, e.g.:

- Missing or invalid encoding value
- Missing or invalid symbology identification

Non-critical Errors

Errors that were caused by not or wrongly defined parameters (e.g. barcode height = 0). While automatically recovering these errors, the relevant parameters are allocated with standard values according to the following listing:

- RES, Standard Value = 150.
- XPOS, Standard Value = 0.
- YPOS, Standard Value = 0.
- HIGH, Standard Value = 10 mm
- MARGIN, Standard Value = 0.
- OFFSET, Standard Value = 0.
- ROT, Standard Value = 0.
- UNIT, Standard Value = 'mm'
- SIGN, Standard Value = 0.
- B, Standard Value = 10,
- BB, Standard Value = 2xB,
- BBB, Standard Value = 3xB
- BBBB, Standard Value = 4xB
- W, Standard Value = B
- WW, Standard Value = 2xB
- WWW, Standard Value = 3xB
- WWWW, Standard Value = 4xB

Input_handling: Determines how the transferred encoding value should be handled.

The permitted values are:

0 The transferred encoding value is not changed

1 Leading zeros are deleted

2 Leading spaces are deleted

3 Leading spaces and zeros are deleted

+10 Add 10 to Input_handling to delete automatically invalid characters from the input string. Example:
Code 39, Encoding '123ä456'. 'ä' is an improper character for this code and will usually cause en error. If you set input_handling to 10, 11, 12 or 13, the 'ä' will be filtered out and only '123456' will be encoded.

Res Resolution of the barcode graphic. The standard value is 150 dpi. Please make sure that the resolution defined in the SETTINGS program has to be congruent with the resolution set in the form with graphic type **BMP** (no so for SAPscript in the **OTF** format). If you wish to print the barcode, make sure that the selected resolution is supported by your printing device. Most laser printers support resolution values of 75, 150, 300 and 600 dpi. Some printers, however, such as a number of ZEBRA printers, only support 202 dpi.

Note: The higher the resolution, the more data are produced and the more computing time per barcode is needed. Although the barcode generation is usually very fast and ranges in the area of milliseconds, this could make itself felt when printing labels with a lot of barcodes per page.

The smaller the resolution, the more erratic the barcode width of the entire barcode rises when changing the module width (see also Chapter 8).

Note: In case of laser printers, the graphic resolution can deviate from the other printing resolutions.

The standard values for all 4 parameters were chosen in such a way that you will only have to change them in very rare cases. As a rule, you can take over these values unaltered.

Note: Sometimes it is necessary to change one of these “global” parameters for just one barcode, e.g. the resolution. In this case, please re-define the resolution in the form routine in which the single barcode characteristics were determined. This setting will then be valid for that special barcode as the processing of the barcode variables is carried out according to the global variables. Do not, however, delete the global settings for such a parameter, as this could lead to problems in case new barcodes need to be defined and you perhaps happen to forget to redefine such a parameter.

9.2 Definition of the Single Barcode Characteristics

The single barcode characteristics are set in a form routine whose name has to be congruent with the value of parameter **BARC_IDENT** from the form. As a standard, 4 such form routines are pre-defined in the ABAP program: **BARCODE01, BARCODE02, BARCODE03 and BARCODE04**. However, you can at any time add further barcode definitions to the SETTINGS program. We will explain the single characteristics according to the parameters that were defined in the form **BARCODE03**:

```
FORM barcode03.
  symbology = '39'.
  w_chksum = ''.
  b = 12.
  barc_high = '13'.
  unit = 'mm'.
  margin = '0.00'.
  offset = '0.00'.
  rot = 0.
ENDFORM.
```

SYMBOLOGY Determines with which barcode symbology the encoding is carried out.

The permitted values are:

'2o5interl'	2 of 5 interleaved
'2o5indust'	2 of 5 industrial
'2o5matrix'	2 of 5 matrix
'2o5gp'	2 of 5 German postal bar code (11 and 13)
'39'	Code 39
'39e'	Code 39 extended
'39f'	French postal 39 A/R
'39d'	Danish 39 PTT
'93'	Code 93
'93e'	Code 93 extended
'codabarXY'	Codabar with the relevant START and STOP character. There are 4 START and STOP characters: „A“, „B“, „C“ and „D“. They can be set at any random sequence. E.g. select **'codabarAC'** in order to use **„A“** as a **START** and **„C“** as a **STOP** character.
'128-A'	Code 128 A
'128-B'	Code 128 B
'128-C'	Code 128 C
'E.A.N.128'	Code EAN128
'128auto'	Code 128 Autoswitch
'ucc-128'	Code UCC-128

'128HIBC'	HIBC Barcode, Single-line, based upon Code 128 for **supplier** and **provider label.** Supplier and provider data have to be separated by a „/“. The sign „+“ at the beginning can but does not have to be set (in this case, it is automatically added by RBarc+). The HIBC check digit is automatically created, therefore must not be transferred as well.
'128HIBCP'	HIBC barcode for provider label. The sign „+“ at the beginning is added automatically. The HIBC check digit is computed automatically.
'MSI1'	MSI (without check digit)
'MSI2'	MSI 10
'MSI3'	MSI10+CHK10
'MSI4'	MSI+CHK11+CHK10
'EAN-13'	EAN/JAN 13
'EAN-13+2'	EAN/JAN 13+2 (with a two-figure add-on)
'EAN-13+5'	EAN/JAN 13+5 (with a five-figure add-on)
'EAN-8'	EAN/JAN 8
'EAN-8+2'	EAN/JAN 8+2 (with a two-figure add-on)
'EAN-8+5'	EAN/JAN 8+5 (with a five-figure add-on)
'UPC-A'	UPC-A
'UPC-A+2'	UPC-A+2 (with a two-figure add-on)
'UPC-A+5'	UPC-A+5 (with a five-figure add-on)
'UPC-E'	UPC-E
'UPC-E+2'	UPC-E+2 (with a two-figure add-on)
'UPC-E+5'	UPC-E+5 (with a five-figure add-on)

.

W_CHKSUM Determines whether the barcode is to be created with or without a check digit. This parameter is only evaluated in case of barcode symbologies permitting an optional check digit. Should the barcode symbology strictly demand a check digit (e.g. Code 128), this parameter will be ignored.

The following barcodes permit an optional check digit:

Code 39

Code 39 extended

Code 2 of 5 interleaved

Code 2 of 5 matrix

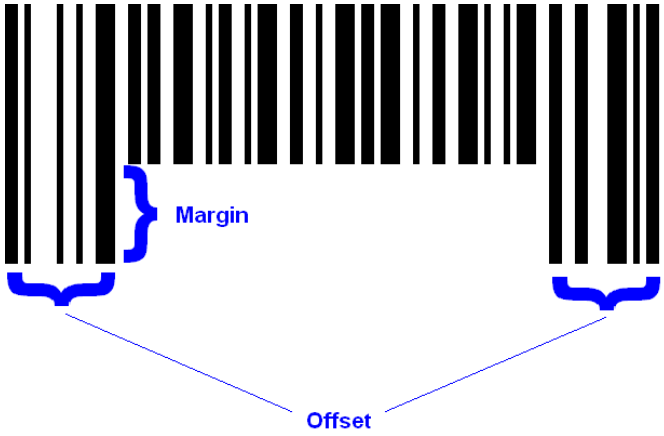
B The so-called module width, being the width of a narrow bar. The width of other bars and all spaces is calculated automatically. Should it be necessary, the other bars and spaces

can be defined manually. This is provided by the parameters „**BB**“, „**BBB**“, „**BBBB**“, „**W**“, „**WW**“, „**WWW**“ and „**WWWW**“. According to the barcode symbology used, there are 2 or 4 different widths (see table in Chapter 8.4). Normally, they are defined in a ratio of 1:2 or 1:2:3:4.
The measurement unit is 1/720 inch. Read more about the barcode width in Chapter 8.3.

Unit Measurement unit for the barcode height (parameter **BARC_HIGH**), the depth of the margin (parameter MARGIN) and the width of the margin next to the margin (parameter OFFSET). Permitted values are „MM“ (millimeter), „CM“ (centimeter) and „INCH“. These values can be written in small as well as in capital letters.

Margin Depth of the margin, should a HRT be wholly or partly embedded into the barcode (see illustration further down). The measurement unit for this is determined by the parameter UNIT.

Offset With this parameter, it is determined how wide the margin left and right of the indentation should be (see illustration). It thus indirectly determines the width of the margin defined with the parameter MARGIN. As the left and the right margins are equal, the indentation of the barcode will always be embedded symmetrically.



Rot Determines whether and how the barcode is rotated. Permitted values are: 0, 90, 180 and 270 (degrees). The rotation is carried out to the right. The value here is absolute, meaning the rotation is always carried out from 0 degrees to the right.

10 More about Barcodes

10.1 Linear Barcode Types

Linear barcodes (also called 1D or one-dimensional barcodes) consist of bars and spaces that are set at a specific ratio by a binding specification. In general, the 1D barcodes are separated into so-called **2-Width** and **4-Width** barcodes.

2-Width barcodes are barcodes consisting of bars and spaces with 2 different widths. One defines a narrow and a wide module – no matter whether it represents a bar or a space (meaning a narrow bar is as wide as a narrow space and a wide bar is as wide as a wide space). In this case, the ratio between narrow and wide has to lie between 1:2 and 1:3. Should the width-ratio be any different, this will – as a rule – lead to an illegible barcode. The following barcodes supported by RBarc+ are **2-Width** barcodes:

2 of 5 interleaved
2 of 5 matrix
2 of 5 industrial
2 of 5 German Postal Barcode
Code 39
Code 39 extended
Codabar
MSI

Example of a 2-Width Barcode Symbol

4-Width barcodes show bars and spaces with 4 different widths. However, also here the bars and spaces of the same type have to be equally wide. The width-ratio has to be 1:2:3:4. The following barcodes supported by RBarc+ are **4-Width** barcodes:

Code 93
Code 93 extended
Code 128-A
Code 128-B
Code 128-C
Code 128-Auto
Code EAN-128
Code UCC-128
Code 128 HIBC
Code 128 HIBCP
Code EAN-13
Code EAN-13+2
Code EAN-13+5
Code EAN-8
Code EAN-8+2
Code EAN-8+5
Code UPC-A
Code UPC-A+2
Code UPC-A+5
Code UPC-E
Code UPC-E+2
Code UPC-E+5

Example of a 4-Width Barcode Symbol

Note: As a rule, you will only have to define the width of the narrowest bar (parameter „**B**“). All other bars and spaces will be automatically computed by RBarc+. With **2-Width** barcodes, the width-ratio will be **1:2** and with **4-Width** barcodes the ratio will be **1:2:3:4,** and each space is calculated in equal width as the bar of the same type. Should you wish to use a different width-ratio for a 2-Width barcode, you have to define your own parameter „BB“. Please make sure, however, to keep the specified ratio between 1:2 and 1:3, otherwise the barcode will be illegible.

10.2 Definition of the Module Width

The module width (the width of the narrowest bar) is set by parameter „**B**“ in program **ZSS_BC_SETTINGS12** (for SAPscript) and with program **ZSF_BC_SETTINGS12** (for SmartForms). The measurement unit is 1/720 inch. This measurement unit is resolution-independent; therefore these settings do not always have to be changed when you decide to change the resolution of the barcode graphic.

10.2.1 Module Width versus Resolution.

As the module width, as mentioned in 8.2, is set in the resolution-independent unit of 1/720 inch, it is recalculated in resolution-dependent units (pixels) during runtime. The result, therefore, will always be an integral figure. For example, the figure 12 with a resolution of 150 dpi calculates 3 pixels for one module and with a resolution of 300 dpi, it is 5 pixels for one module. The calculation formula is: Br = (B x Resolution) / 720. The result up to X.5 is rounded up and from X.5 it is rounded down.

Note: Owing to the above-described facts, it is possible that the alteration of parameter „**B**“ is not entirely effective on the allover width of a barcode in case of a certain resolution. If, for example, B = 12, the result with 150 dpi is 12 * 150 / 720 = 2.5 and – rounded up – 3 pixels. Should the value of **B** be increased to 15, this results in 15 * 150 / 720 = 2.9, and that still are 3 pixels, rounded up. In case of a resolution of 300 dpi, things look entirely different, as 12 * 300 / 720 = 5 pixels and 15 * 300 / 720 = 6.25, rounded-down, that are 6 pixels. That way in case of 150 dpi, changing parameter „**B**“ from 12 to 15 would have no effect on the barcode width, but with 300 dpi, there would be an effect.

10.3 Determination of the Allover Width of the Barcode

Now, we will try and find out in what way parameter „**B**“ should be selected if a specific barcode allover-width is required. In this case, the procedure is as follows:

- Define parameter „**B**“ with a value of, for example, **10**.
- Print out the barcode and measure its allover-width.
- Determine the ratio between the measured allover-width and the requested allover-width.
- Increase or decrease parameter „**B**“ by the calculated factor.
- Re-print the barcode.

Note: Due to the resolution and the fact that, for quality reasons, a barcode cannot simply be stretched apart, the allover-width of the barcode erratically increases or decreases (when, for example, a barcode consists of 50 bars and you increase parameter „**B**“ in such a way that the bar is widened by 1 pixel, then the allover-width is increased by over 100 pixels, as also the spaces increase by one pixel each and the wide modules even by 2 pixels).

Note: Should you be unable to reach the requested barcode allover-width in case of a specific resolution, please increase the resolution of the barcode graphic (parameter „**RES**“). That way, the erratic leaps will get smaller, because the higher the resolution, the smaller one pixel will be.

10.4 Encodable Symbols and Input Format

Each barcode symbology possesses a so-called encodable character set. This character set includes all characters and signs that are encodable with a certain barcode symbology. The attempt to encode a character with a symbology not included in this character set will lead to a critical error. Therefore, please check before implementing a barcode solution for your SAP system which characters will have to be encoded and select the corresponding symbology. An example for such a problem would be if figures followed by figures after the decimal comma were to be encoded with Code 39. As the decimal comma does not belong to the character set of Code 39, a critical error will occur. A solution in this case would be to replace the comma with a decimal point (this point belongs to the character set of Code 39), or to select the symbology Code 39 Extended or 128-Autoswitch; their character sets also include the decimal comma. Nevertheless, you will have to exercise special care as a symbology-change should be agreed upon by all partners in the logistics-chain, otherwise it could lead to unpleasant surprises when, for example, a barcode-reader used by the goods recipient cannot read the new barcode.
Some barcode symbologies also require input in a very specific format. With Code EAN-13, for example, exactly 12 figures have to be transferred – not more and not less. If this rule is not kept, this will also lead to a critical error. These rules you will find in the table included in the following chapter.

10.4.1 Character Set of the Barcode Symbologies Supported by RBarc+

In the following table, the character set of all barcode symbologies supported by RBarc are listed. You will also find additional information on the symbology types and whether or not a check digit is calculated optionally or obligatory. "**YES**" in the column "Check Digit" means, that the check sum is mandatory and the parameter **w_chksum** will be ignored. Also at "**NO**" the parameter w_chksum will be ignored, but in this case no check sum will be calculated (because not provided). The value "**OPTIONAL**" means, that the parameter w_chksum will be analyzed and the check sum calculated, if it was set to w_chksum = '**X'**.

Bar Code Name	Character Set	Check Digit	Type	Comment
2 of 5 interleaved	00 – 99	OPTION	2-Width	Only pairs of figures permitted
German Postal Identcode	11 figures fix	Yes	2-Width	Setup as in 2 of 5 interleaved
German Postal Leitcode	13 figures fix	Yes	2-Width	Setup as in 2 of 5 interleaved
2 of 5 industrial	0-9	OPTION	2-Width	
2 of 5 matrix	0-9	OPTION	2-Width	
Code 39	0-9 A - Z The sign '-' (Minus) The sign '.' (Dot) The space character The sign '$' The sign '/' The sign '+' The sign '%'	OPTION	2-Width	
Danish PTT 39 Barcode	10 figures fix	Yes	2-Width	
French Postal 39 A/R	RA + 8 figures RB + 8 figures	Yes	2-Width	
Code 39 extended	ASCII 0x00 – 0x7F	OPTION	2-Width	Very wide, creates 2 symbols per character, e.g. a=+A
Codabar	0-9	Yes	2-Width	4 different start and stop characters can be defined by the user.

Bar Code Name	Character Set	Check Digit	Type	Comment
MSI	0-1	No	2-Width	
MSI (Chk 10)	0-1	Yes	2-Width	
MSI (Chk 10 10)	0-1 10 10	Yes	2-Width	2 Check Digits
MSI (Chk 11 10)	0-1 11 10	Yes	2-Width	2 Check Digits
Code 93	0-9 A-Z The sign '-' (Minus) The sign '.' (Dot) The space character The sign '$' The sign '/' The sign '+' The sign '%'	Yes	4-Width	2 Check Digits
Code 93 extended	ASCII 0x00-0x7F	Yes	4-Width	2 Check Digits. Very wide, creates 2 symbols per character, e.g. a=+A
Code 128 A	ASCII 0x00-0x5F FNC1 FNC2 FNC3 FNC4	Yes	4-Width	
Code 128 B	ASCII 0x20-0x7E DEL FNC1 FNC2 FNC3 FNC4	Yes	4-Width	
Code 128C	00-99	Yes	4-Width	2 Check Digits. Very wide, creates 2 symbols per character, e.g. a=+A
Code 128 Autoswitch	ASCII 0x00-0x7F	Yes	4-Width	Changes automatically between 128A, 128B and 128C
UCC-128	19 figures fix	Yes	4-Width	FNC1 is automatically put in after the start character
EAN128	0-9	Yes	4-Width	FNC1 is automatically put in after the start character
EAN/JAN 8	7 figures fix	Yes	4-Width	
EAN/JAN 8 +2	9 figures fix	Yes	4-Width	with 2-Digit Add-on
EAN/JAN 8+5	12 figures fix	Yes	4-Width	with 5-Digit Add-on
EAN/JAN 13	12 figures fix	Yes	4-Width	
EAN/JAN 13 +2	14 figures fix	Yes	4-Width	with 2-Digit Add-on
EAN/JAN 13+5	17 figures fix	Yes	4-Width	with 5-Digit Add-on
UPC-A	11 figures fix	Yes	4-Width	
UPC-A +2	13 figures fix	Yes	4-Width	with 2-Digit Add-on
UPC-A+5	16 figures fix	Yes	4-Width	with 5-Digit Add-on
UPC-E	7 figures fix	Yes	4-Width	
UPC-E +2	9 figures fix	Yes	4-Width	with 2-Digit Add-on
UPC-E+5	12 figures fix	Yes	4-Width	with 5-Digit Add-on

11 Barcode Setting – A short Summary

In this chapter, you will find a summary of all parameters able to have an effect on the barcode characteristics.

All barcode settings are run in the ABAP programs **ZSS_BCSETTINGS12** (for SAPscript) and **ZSF_BC_SETTINGS** (for SmartForms).

For each barcode, the barcode characteristics are integrated into one form routine. The name of the form routine, e.g. **FORM BARCODE01**, has to be congruent with the value of the parameter **BARC_IDENT**. The parameter **BARC_IDENT** is determined within the form itself.

The following parameters will have an impact on the barcode:

Symbology Barcode symbology (e.g. Code 39 or Code 128A).

W_chksum Calculation of the check digit in case of barcodes that do not necessarily demand a check digit (e.g. Code 39).

B Width of the module in units of 1/720 inches. This indirectly determines the entire barcode width.

Unit Measurement unit for the barcode height, for its margin and the offset.

Margin Depth of the indentation at the bottom margin of the barcode, meant for inserting the Human Readable Text (HRT).

Offset Width of the margins, in case an indentation was defined using the parameter “Margin”.

Rot Barcode rotation in steps of 90 degrees.

Res Resolution of the barcodegraphic.

The exact description of the parameters can be found in **Chapter 7.2**.

There are some further parameters that can be utilized to control the program performance in some cases.

error_handling Determines, how the program reacts should an error occur. For example, an automatic error recovery can be made in cases of so-called non-critical errors (e.g. a negative margin).

Input_handling Determines the handling of the value to be encoded. As a rule, the value will be adopted unaltered. However, with this parameter, it can be determined that, for example, leading zeros or invalid characters must be removed before the barcode generation.

The exact description of the parameters can be found in **Chapter 7.1**

12 Characteristics of the Barcode 128FREE

The RBarc barcode 128FREE is a special implementation of the Code 128 that enables the encoding of characters that cannot be entered via the keyboard.
The Code 128 consists of 3 so-called character sets. The character set „A“ allows, for example, the encoding of the characters with the ASCII-values 0 to 127. The characters 0 to 31, however, cannot be entered via the keyboard. Furthermore, the Code 128 provides the special characters FNC1, FNC2, FNC3 and FNC4 that also do not correspond to any keyboard character. Therefore, a special handling of these characters was implemented in RBarc that enables the developer to make his own compilation of the characters required for a Code 128. Of course, also in such a case the check digit is calculated automatically.

The Code 128 supports the following non-displayable characters:

[NUL], [SOH], [STX], [ETX], [EOT], [ENQ], [ACK], [BEL], [BS], [HT],
[LF], [VT], [FF], [CR], [SO], [SI], [DLE], [DC1], [DC2], [DC3],
[DC4], [NAK], [SYN], [ETB], [CAN], [EM], [SUB], [ESC], [FS], [GS],
[RS], [US],
[FNC1], [FNC3], [FNC2], [FNC4],
[CODE A], [CODE B], [CODE C], [SHIFT],

For encoding all characters of a Code 128 that contains non-displayable characters the form routine „**collect_128**“ is responsible. In this case, the following rules apply:

- All characters to be encoded (also displayable characters) have to be processed with this form routine.
- Displayable characters can be transferred to the form routine in inter-connected groups.
- Non-displayable characters have to be connected to the form routine in single form.
- All characters have to be typecast with the variable „DataType“: Displayable characters with ‚T‘ and non-displayable characters with ‚F‘.
- Start characters and all possibly ensuing changes of the character set between „A“, „B“ and „C“ have to be entered manually by the programmer.

Example:

The encoding of the following character string is to be performed (spaces and square brackets are serving clarification purposes, only)

[START B] [FNC1] 241GA1 [FNC1] 101274 [FNC1] [CODE C] 370587

Implementation:

```
encoding = 'Start Code B'.
DataType = 'F'.
perform collect_128 using encoding DataType.

encoding = 'FNC1'.
DataType = 'F'.
perform collect_128 using encoding DataType.

encoding = '241G1A'.
DataType = 'T'.
perform collect_128 using encoding DataType.

encoding = 'FNC1'.
DataType = 'F'.
perform collect_128 using encoding DataType.

encoding = '101274'.
DataType = 'T'.
perform collect_128 using encoding DataType.
```

```
encoding = 'FNC1'.
DataType = 'F'.
perform collect_128 using encoding DataType.

encoding = 'Code C'.
DataType = 'F'.
perform collect_128 using encoding DataType.

encoding = '370587'.
DataType = 'T'.
perform collect_128 using encoding DataType.
```

Should the barcode type “128FREE” be generated, the variable “encoding” can remain blank. RBarc automatically recognizes the barcode type and does not interpret the variable „encoding“, instead it processes the table that was used to create the form routine „collect_128“.

13 The Barcode GS1-128

The RBarc barcode GS1-128 is the official successor of the barcode EAN-128.
In order to be clearly distinguished from other codes of type Code 128, he encodes the special characters "FNC1" on the first place after the start. "FNC1" is not displayable character, however, is transmitted by the laser scanner, so that the processing software can check whether it is indeed a GS1-128 barcode.

For the barcode GS1-128 the symbology '128GS1' was implemented in RBarc+. For this symbology the special character 'FNC1' is always generated automatically in the first place after the START, so it may not be explicitly specified.

According to the specification for the barcode GS1-128 fields may be separated by special characters, such as "FNC1" or "GS1". Since these special characters can not be entered via the keyboard, a special method was developed that allows you to encode these special characters with RBarc+.
For this reason all encoding data has to be classified passed to the form routine "collect_128GS1". A distinction is made between two types of data: text and function code. As long as data to be encoded is data, which can also be entered via the keyboard (which are ASCII characters from 32 dec. To 126 dec.), they are declared as text variables. This is done with the statement **DataType = 'T'.** Text variables can have arbitrary lengths. If, in contrast to characters that can not be entered using the keyboard, the encoding data cannot be entered using the keyboard, then such data must be declared as a function codes. This is done with the statement **DataType = 'F'**. Function codes(eg '**GS**' may only be coded individually, ie if two function codes should stand behind each other, then the form routine must be called collect_128GS1 two times.

The procedure described above applies for SAPscript, Smart Forms and Interactive Forms.

Example:

The material number and the number of its units must be encoded, separated by the function code "GS" (Group Separator).

Implementation for Smart Forms and Interactive Forms:

```
encoding = matnr.
datatype = 'T'.
perform perform collect_128gs1  in program zsf_bc_settings12
using encoding DataType.

encoding = 'GS'.
datatype = 'F.
perform perform collect_128gs1  in program zsf_bc_settings12
using encoding DataType.

encoding = meng.
datatype = 'T'.
perform perform collect_128gs1  in program zsf_bc_settings12
using encoding DataType.

BARC_IDENT = 'BARCODE07'.
encoding = 'dummy'.
PERFORM GEN_BARCODE IN PROGRAM ZSF_BC_SETTINGS12
USING encoding barc_ident
CHANGING barc_name checksum encoding_return.
```

Remark:

- The code like in the example above has to be inserted into the code node of the form.
- It is assumed that **matnr** and **meng** are determined from the database.
- Since the implementation of GS1-128 is backwards compatible with earlier versions of RBarc+, the variable "encoding" - although it is not required - has to be filled with the value 'dummy', before it is passed to GEN_BARCODE.
- Use ZAF_BC_SETTINGS12 for Interactive Forms in place of ZSF_BC_SETTINGS12

Implementation in SAPscript:

```
DEFINE &ENCODING& = &MATNR&
 DEFINE &DATATYPE& = 'T'
 PERFORM COLLECT_128GS1 IN PROGRAM ZSS_BC_SETTINGS12
  USING &ENCODING&
  USING &DATATYPE&
 ENDPERFORM

 DEFINE &ENCODING& = &GS&
 DEFINE &DATATYPE& = 'F'
 PERFORM COLLECT_128GS1 IN PROGRAM ZSS_BC_SETTINGS12
  USING &ENCODING&
  USING &DATATYPE&
 ENDPERFORM

 DEFINE &ENCODING& = &MENG&
 DEFINE &DATATYPE& = 'T'
 PERFORM COLLECT_128GS1 IN PROGRAM ZSS_BC_SETTINGS12
  USING &ENCODING&
  USING &DATATYPE&
 ENDPERFORM

 DEFINE &BARC_IDENT& = 'BARCODE07'
 DEFINE &ENCODING& = 'DUMMY'
 DEFINE &XPOS& = '10.00'
 DEFINE &GRAPH_TYPE& = 'OTF'

PERFORM GEN_BARCODE IN PROGRAM ZSS_BC_SETTINGS12
  USING &BARC_IDENT&
  USING &ENCODING&
  USING &GRAPH_TYPE&
  USING &XPOS&
  CHANGING &BARC_NAME&
  CHANGING &USED_LINES&
  CHANGING &ENCODING_RETURN&
  CHANGING &CHECKSUM&
ENDPERFORM
```

Bemerkungen:

- The code like in the example above must be inserted into the SAPscript form.
- It is assumed that **matnr** and **meng** are determined from the database.
- Since the implementation of GS1-128 is backwards compatible with earlier versions of RBarc+, the variable "encoding" - although it is not required - has to be filled with the value 'dummy', before it is passed to GEN_BARCODE.

14 Index