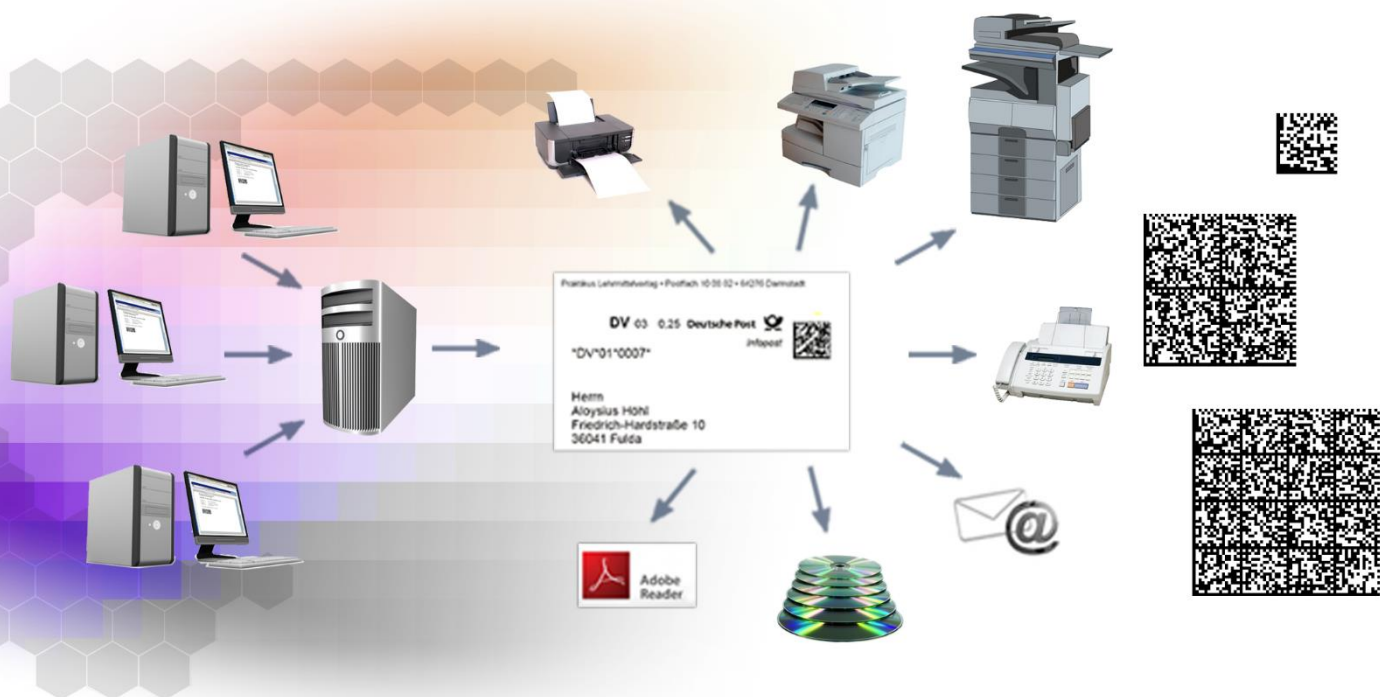




SUCHY MIPS



RBarc/Datamatrix For SAP® Systems Version 13 Reference Manual

Release: December 2021

The information contained in this document is subject to change without notice.

Suchy MIPS makes no representations of warranties either express or implied, with respect to this publication and accompanying software and specifically disclaims any implied warranties with regard to its sales potential or fitness for any particular purpose.

© Copyright Suchy MIPS 2007-2021
All rights reserved.

No parts of this publication may be reproduced, transmitted, stored in a retrieval system, or translated into any human or computer language, in any form by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of the copyright owner, Suchy MIPS in Munich/Germany. Copyright infringement carries with it serious civil and criminal penalties under international copyright laws.

This manual describes the installation and use of copyrighted software. Said software is licensed to the End User for use only in strict accordance with the End User License Agreement. Licensee is advised to read this End User License Agreement carefully before commencing use of product.

R/3, SAP, mySAP and Netweaver are trademarks of SAP® AG.
MS-Windows is a trademark of Microsoft® Corporation, Inc.
All other product names are trademarks of their respective holders.

Suchy MIPS GmbH
Prinzregentenstrass 128

81677 Munich
Germany

Phone: +49 (0) 89 - 944 19 77 - 0
Fax: +49 (0) 89 - 944 19 77 - 13
e-mail: info@suchymips.de
Internet: www.suchymips.de

Contents

No.	Title	Page
1	Introduction	3
2	System requirements.....	4
3	Installation.....	5
4	Differences between the full and the demo version of RBarc/Datamatrix.....	6
5	Testing the installation of RBarc/Datamatrix	7
5.1	Testing with SAPscript.....	7
5.2	Test with Smart Forms	8
5.3	Testing with Interactive Forms.....	9
6	Embedding the Datamatrix bar code in forms	10
6.1	The functional principle of RBarc/Datamatrix.	10
6.2	Categorization of the variables	11
6.3	Categorization of the variables	12
6.4	Datamatrix printing with SAPscript	14
6.4.1	Bar code definition in a SAPscript form	14
6.5	Datamatrix printing with Smart Forms	18
6.5.1	Bar code definition in a Smart Forms form.....	18
6.6	Datamatrix printing with Interactive Forms	24
6.6.1	Enhancement of the Interactive Forms Interface	24
6.6.2	Enhancement of the Interactive Forms Form	29
6.7	Definition of bar code properties.....	32
6.8	Job	42
7	Example of a job definition for a Smart Forms Form.....	44
7.1	Job.	44
8	Annex 1: Creating a new package (development class).	46
9	Annex 2: Creating an include in the ABAP Workbanch	48
10	Annex 3: Creating a program in the ABAP workbanch.	51
11	Annex 4: inserting content of installation files into an ABAP Program.	54
12	Annex 5: Activating of programs and includes	55
13	Anhang 6: Executing of an ABAP program.	57
14	Annex 7: Importing of a SAPscript form	58
15	Annex 8: Uploading of a Smart Forms Form.....	60
16	Annex 9: Testing a Smart Forms form.....	62
17	Annex 10: Upload an Interactive Forms Interface.	64
18	Annex 11: Upload an Interactive Forms Form.....	66
19	Annex 12: Test an Interactive Forms Form	67

1 Introduction

RBarc/Datamatrix is an ABAP Program for generating the 2D bar code Datamatrix on SAP Systems (R/3, MySAP, ERP, Netweaver...). This functionality is available for **SAPscript** and **Smart Forms** and **Interactive Forms**.

RBarc/Datamatrix is fully compliant with the GS1 DATAMATRIX ECC 200 specification.

RBarc/Datamatrix generates the 2D bar code in such a way, that it becomes an integral part of the document. The resulting advantages are that documents with bar codes can be printed on any printer - regardless of the manufacturer and the printer language used. Hardware extensions are not required. Another advantage is that bar codes are always visible on the documents, even if the document is faxed, emailed, converted to a PDF document or has been archived.

Important:

RBarc/Datamatrix does NOT change the SAP standard. **RBarc/Datamatrix** programs and includes begin with the letter "Z" and will be saved in the customers area. Thus, updates will not overwrite any parts of **RBarc/Datamatrix**.

2 System requirements

RBarc/Datamatrix requires an installed SAP® System R/3 vers. 3.x or higher (also MySAP ERP and NetWeaver).

Important:

RBarc/Datamatrix is dedicated to system administrators (installation) and developers (bar code configuration and embedding in the form). Thus, a good knowledge in SAP basis is required for installation and a good knowledge in **SAPscript** or **Smart Forms Interactive Forms** (up to the used technology) is required for embedding bar codes in the appropriate forms. A good knowledge in ABAP will be helpful for the bar code configuration.

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

ACTVT = 01, 02, 03, 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 bar code will appear on the output or generated barcodes will not be deleted.

3 Installation

The installation steps will be described here in a short form. If you are not familiar with some operations, (eg. creating of a new program), click on the blue link next to the description. This will bring you to the page with the more detailed description of the operation.

The program consists of an ABAP program (report) and an ABAP include. Both files are located in the "Install" directory.

The test forms for **Smart Forms**, **SAPscript** and **Interactive Forms** are located in corresponding subdirectories: "Test-Smartforms", "Test-SAPscript", "Test-Interactiveforms".

1. Start the **SAP GUI** and logon at the SAP System as Administrator.
2. Start the **Object Navigator** (transaction **SE80**) and create the package (development class) **ZDATAMATRIX** (recommended).
(go to page 46: [Annex 1: Creating a new package \(development class\)](#))
3. Create an include **ZDATAMATRIX** in the package (development class) **ZDATAMATRIX** and delete its content, which was generated automatically.
(go to page 48: [Annex 2: Creating an include in the ABAP Workbench](#))
4. Copy the content from the file **ZDATAMATRIX.INC** and paste it into the include **ZDATAMATRIX**. Save the include and activate it.
(go to page 54: [Annex 4: inserting content of installation files into an ABAP Program](#))
5. Create a program (report) **Z_SET_DATAMATRIX** in the package **ZDATAMATRIX** and delete its content, which was generated automatically.
(go to page 46: [Annex 3: Creating a program in the ABAP workbench](#))
6. Copy the content from the file **Z_SET_DATAMATRIX.PRG** and paste it into the report **Z_SET_DATAMATRIX**. Save the report and activate it.

This completes the installation of RBarc/Datamatrix.

Now test objects can be installed to print one Smartforms, Interactiveforms and Sapscript form each with a Datamatrix barcode.

7. Create a program (report) **ZSF_DM_PRINT** in the package **ZDATAMATRIX** and delete its content, which was generated automatically.
8. Copy the content from the file **ZSF_DM_PRINT.PRG** and paste it into the report **ZSF_DM_PRINT**. Save the report and activate it.
9. Create a program (report) **ZIF_DM_PRINT** in the package **ZDATAMATRIX** and delete its content, which was generated automatically.
10. Copy the content from the file **ZIF_DM_PRINT.PRG** and paste it into the report **ZIF_DM_PRINT**. Save the report and activate it.
11. Create a program (report) **ZSS_DM_PRINT** in the package **ZDATAMATRIX** and delete its content, which was generated automatically.
12. Copy the content from the file **ZSS_DM_PRINT.PRG** and paste it into the report **ZSS_DM_PRINT**. Save the report and activate it.

(go to page 55: [Annex 5: Activating of programs and includes](#))

4 Differences between the full and the demo version of RBarc/Datamatrix

- The maximum matrix size in the demo version is 14 x 14 (parameter msize = 3). This means, that 8 codewords may be encoded. 16 digits or 10 characters may be encoded with 8 Codewords. In mixed mode, or when using special characters, such as GS (Group separator) the amount of encryptable data is not easily predictable, since the change from one alphabet to another alphabet creates a certain overhead data the amount of which depends not only on the alphabets used but also on the distribution of characters.
- The number „1“ will be spontaneously exchanged against the „0“.

Important:

If you want to test with real data, contact us in order to negotiate a test phase with the full product version. In the chargeable test phase user can test for 4 weeks with the full product version having an appropriate support.

5 Testing the installation of RBarc/Datamatrix

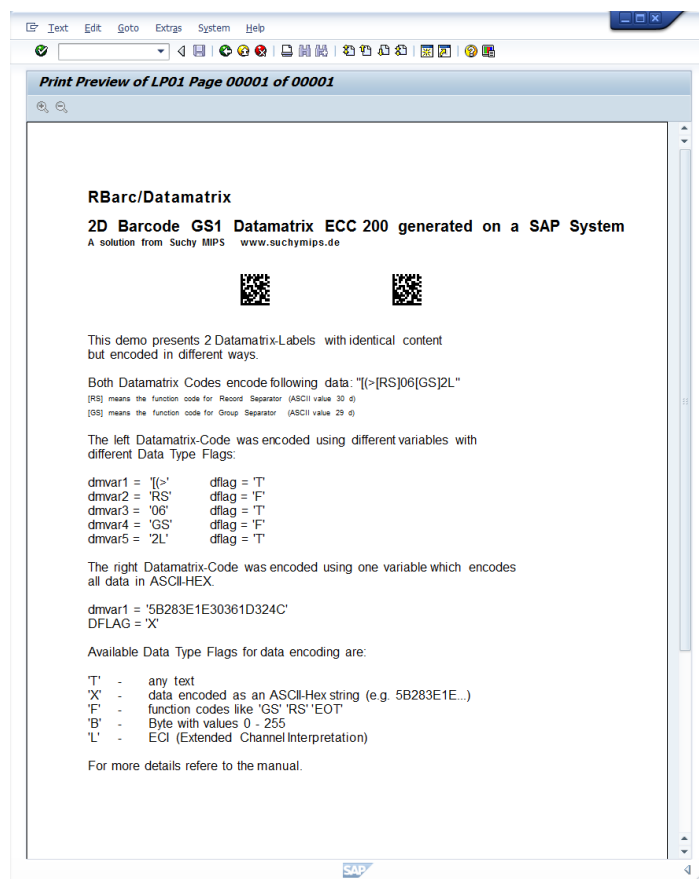
The Test-Smartforms, Test-Interactiveforms and Test-Sapscript directories contain test form objects and associated print programs (reports).

5.1 Testing with SAPscript

1. Execute the standard program **RSTXSCR**P and import the **SAPscript** Form **ZSS_DM_FORM** from the file **ZSS_BC_FORM.FOR**.
(go to page 58: [Annex 7: Importing of a SAPscript form](#))
2. Execute the report **ZSS_DM_PRINT** to print the **SAPscript** test form **ZSS_BC_FORM**.
(go to page 57: [Annex 6: Executing of an ABAP program](#)).

The program **ZSS_PRINT** prints **SAPscript** Form **ZSS_DM_FORM**.

You can see the datamatrix bar code already in the print preview. It should look similar to this:



To embed a bar code in your own SAPscript Form, go to page 14: [Bar code definition in a SAPscript form](#))

5.2 Test with Smart Forms

1. Start the transaction „**Smartforms**“ and upload the **Smart Forms** Form **ZSF_DM_FORM** from the file **zsf_dm_form.xml**. Save and activate the form.
(go to page 60: [Annex 8: Uploading of a Smart Forms Form](#))

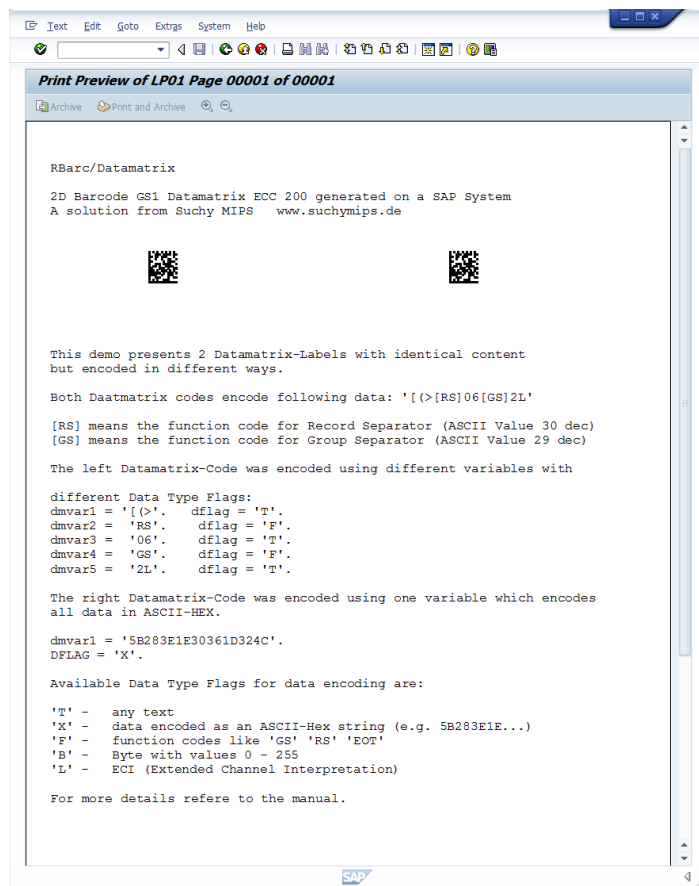
Notice:

This function is available beginning with R/3 vers. 4.7. User of R/3 vers. 4.6x must create a **Smart Forms** Form manually. Go to page 18: [Datamatrix printing with Smart Forms](#)

1. Execute the report **ZSF_DM_PRINT** to print the **Smart Forms** test form **ZSF_BC_FORM**.
(go to page 57: [Annex 6: Executing of an ABAP program](#)).

The program **ZSF_PRINT** prints **Smart Forms** Form **ZSF_DM_FORM**.

You can see the datamatrix bar code already in the print preview. It should look similar to this:

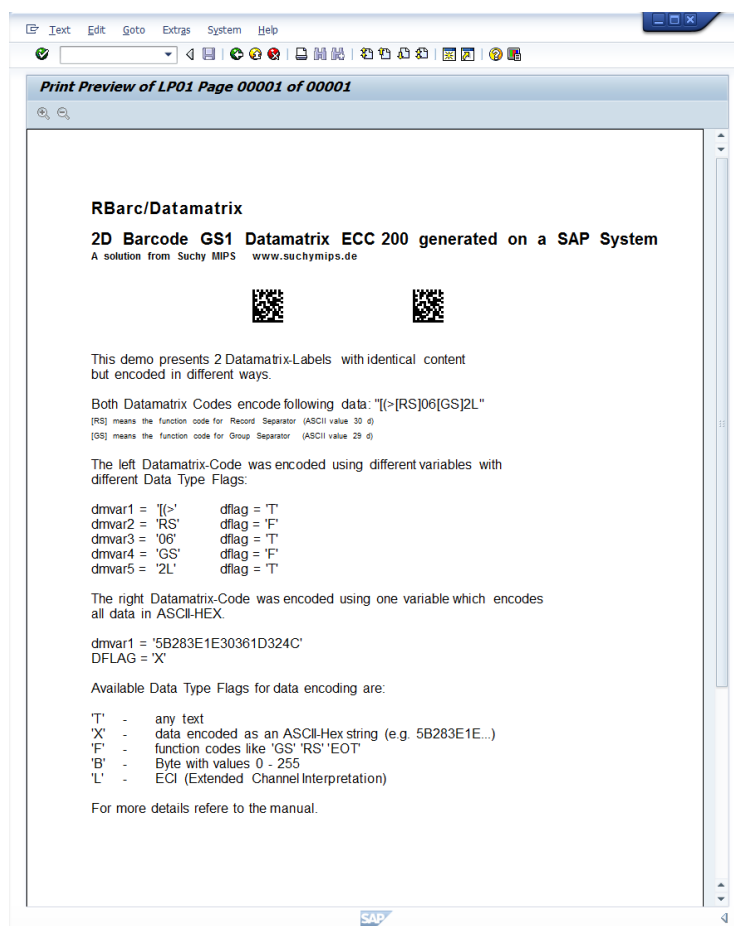


To embed a datamatrix bar code in your own **Smart Forms** form, go to page 18: [Datamatrix printing with Smart Forms](#)

5.3 Testing with Interactive Forms

1. Start the transaction „**SFP**“ and upload the **Interactive Forms** interface **ZIF_DM_INTERFACE** from file **zif_dm_interface.xml**. Save and activate the interface.
(go to page 64: [Annex 10: Upload an Interactive Forms Interface](#))
2. Start the transaction „**SFP**“ and upload the **Interactive Forms** form **ZIF_DM_FORM** from the file **zif_dm_form.xml**. Save and activate the form.
(go to page 66: [Annex 11: Upload an Interactive Forms Form](#))
3. In order to print the **Interactive Form** test form **ZIF_DM_FORM**, open the form and select from the menu „Form / Test“.
(go to page 67: [Annex 12: Test an Interactive Forms Form](#)).

You can print out the form on the printer or look at it in the print preview. The result should present itself approximately as follows:



In order to integrate a bar code into its own **Interactive Forms** form, please read further down under: „Printing Bar codes from **Interactive Forms**“ (go to page 24: [Datamatrix printing with Interactive Forms](#)).

6 Embedding the Datamatrix bar code in forms

Datamatrix is a 2D bar code, which can encode lots of data (up to 3116 numeric, 2335 alphanumeric or 1556 byte data). The requirement of ERP applications is not to encode continuous text but much more specific data from data fields of the data base. These fields shall mostly be encoded in one datamatrix label, separated by field separators. Often the data container to be encoded gets a header and a trailer. Below is the working principle of RBarc /Datamatrix as well as the rules for the implementation of Datamatrix Bar codes in the various SAP form technologies: SAPscript Smarto Frms and Interactive Forms.

Useful examples of tasks from the field and appropriate implementations see the chapter:

Example of a job definition for a SAPscript Form

Example of a job definition for a Smart Forms Form

6.1 The functional principle of RBarc/Datamatrix.

RBarc/Datamatrix functions as follows:

- All variables that shall to be encoded in the Data Matrix bar code, must be categorised and written into an interface table in pairs (variable / type). The types of the variables are described in the next section.
- Optional parameters for bar code properties such as **module size**, **matrix size** or **graphic resolution** can be also written into the interface table. The optional parameters are described in the next section over . If optional parameters are not defined, then **RBarc/Datamatrix** will automatically set default values for these parameters.
- The interface table with encoding variables and their corresponding variable type as well as possible optional parameters will be passed to a form routine in the report **Z_SET_DATAMATRIX**. The name of the form routine is different for each SAP form technologie:

for SAPscript:	GEN_DATAMATRIX_SS
for Smart Forms:	GEN_DATAMATRIX_SF
for Interactive Forms:	GEN_DATAMATRIX_IF

- At runtime the program **Z_SET_DATAMATRIX** generates the bar code dynamically and passes the unique name of the created bar code or the bar code as a binary object back to the form.
- The bar code will be included in the form dynamically.
- Finally, the bar code will be deleted from the system.

Notice

The transfer of all parameters via an interface table allows encoding of any number of variables, without each time the interface should be adapted.

The interface between the form and the report **Z_SET_DATAMATRIX** is, in principle, for all SAP form technologies the same. However, since form technologies are different, there are differences between SAPscript, Smart Forms and Interactive Forms in implementation of the interface. Therefore, the implementation details are described in the appropriate chapters for each form technology.

6.2 Categorization of the variables

The bar code Datamatrix can encode all ASCII values in the range 0 – 255 (dec). Because not all of these values can be represented as printable characters, we have developed a special procedure, which allows to generate each value of the full ASCII range. For this reason, each variable must be categorized, using the parameter "**dflag**" before it will be saved in the interface table record. The following categories of variables (values for "**dflag**") are valid:

'T' – each alphanumeric character, eg. 'ABCDabcd12345AbCd'.

'X' – variable in ASCII HEX format 0x00 – 0xFF. Example: the variable '**123ABC**' with **dflag** = '**T**' could also be encoded as the value '**303132414243**' using dflag = '**X**'. Using this method all values 00 – 255 can be encoded, even non printable characters like the binary value 0 as ASCII-HEX '00'.

'Z' – variable in ASCII HEX format 0x00 – 0xFF. All values are encoded in the so-called Base 256 mode (1 codeword per byte).

'Y' – variable in ASCII format. All characters are coded in the so-called Base 256 mode (1 codeword per byte).

'F' – variable is a function code. Only one such variable can be defined per step. If more then one such variable should be encoded, each one has to be saved separately together with its category (dflag) as a record in the interface table. Following function codes are valid:

- ASCII characters 0 - 31:
'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'.
- The EAN function character: '**FNC1**'
- The "Extended Channel Interpretation" character: '**ECI**'
- Macro 05 for the industrial Header/Trailer: [(>RS05GS RSEOT: '**MAC05**' .
- Macro 06 for the Industrial Header/Trailer [(>RS06GS RSEOT: '**MAC06**'.

'B' – variable is a decimal binary value. Values 0 – 255 are valid. Only one value per step is allowed. If more then one such variable should be encoded, each one has to be saved separately together with its category (dflag) as a record in the interface table. This method also can be - similar to type 'X' use for encoding all ASCII characters.

'L' – the variable is an "Extended Channel Interpretation" value. Each variable has to begin with 'ECI', followed by a 6 digits number. Valid ECI numbers are 0 - 999999. The default value is 000003 (ASCII for chars 0 - 127 and ISO 8859-1 for chars 128 – 255). ECI forces a special character interpretation (default = western Europe). Please remember, that ECI can be interpreted only by special scanners. A detailed description of the ECI protocol can be found in the document "*Extended Channel Interpretation (ECI) Assignments*". Example of an ECI: ECI000978.

6.3 Categorization of the variables

Optional parameters are used to control external bar code properties. These include the **size of the bar code**, the **size of the matrix** and the **resolution of the graphics**. If a parameter is not defined, then it is automatically set to the default value.

MSIZE Size of the matrix in accordance with ISO specification. The ISO standard specifies 30 occurrences of the data matrix bar codes that can be selected with **msize** here. Normally we try to bring the data to be encoded in a matrix as small as possible. In some applications, however, it is necessary to obtain a constant bar code size. In this case, a value for **msize** must be specified. The value should be chosen so that all eligible data have space in the chosen matrix. If it turns out during the run time that the chosen matrix size is too small, an error message is issued and the bar code is not generated.

Permissible values: 0 - 30.

Default value: **0** (automatical calculation of smallest possible matrix).



msize = 4 (16x16)



msize = 11 (36x36)

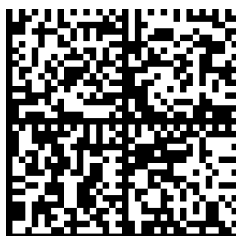
Both bar code encode the same text "Suchy MIPS"

A complete list of all matrix sizes, see the chapter "Definition of the bar code properties"

RES Specifies the resolution at which the bar code graphic is generated. The smaller the resolution, the faster the bar code is generated and the less data are generated. Doubling the resolution with the same bar code size represents a quadrupling of the data quantity or a decrease in the bar code size by four times at constant **X_DIM**. Since the modules of a data matrix bar codes are squares, the resolution has no decisive influence on its quality. Higher resolutions should only be selected if desired bar code size can not be reached with the default resolution. If **RES** is increased without **X_DIM** is changed, then the bar code and its size can be precisely controlled with reduced **X_DIM**. If a higher resolution for the same bar code size is desired, then the parameter **X_DIM** must be increased by the same factor as the resolution.

Permissible values: > 0 (Dots per inch)

Default value: **150**



RES = 300 dpi, X_DIM = n



RES = 600 dpi, X_DIM = n

Both bar code encode the same text "Suchy MIPS"

X_DIM Horizontal size of the smallest module. The larger **X_DIM** the greater the bar code. The unit is 1/RES inches. The higher the resolution the smaller the **X_DIM** unit and the finer the bar code size can be determined.

Permissible values: 1 bis 100.

Default value: **4**



X_DIM = 10



X_DIM = 20

Both bar code encode the same text "Suchy MIPS"

Y_DIM Should be defined only in exceptional cases.

Permissible values: 1 bis 100

Default value: **X_DIM**

XPOS **XPOS** applies only to SAPscript. The parameter determines the horizontal position of the bar code graphic relative to the left border of the window in which the bar code is included. The value must not be negative.

Permissible values: 0 to width of page

Default value: 0

YPOS **YPOS** applies only to SAPscript. The parameter determines the vertical position of the bar code graphic relative to the row where the barcode is included. The value must not be negative.

Permissible values: 1 to length of page

Default value: 0

Autoheight Type of place reservation for the barcode graphic in **SAPscript** forms. If Autoheight is set to 'Y', then the whole space is required on the left and right of it for the graphics and no text can be placed next to it. If the value is set to 'N', then the graphics claims no place.

Permissible values: 'Y' 'N'

Default value: 'N'

6.4 Datamatrix printing with SAPscript

6.4.1 Bar code definition in a SAPscript form

The following example shows the implementation of a data matrix Label with 5 different variables in a SAPscript form:

```
/E ITEM_LINE
* <HL>RBarc/QR-Code
*
* <HL>2D Barcode QR-Code Model 2 generated on a SAP System
* <HN>A solution from Suchy MIPS www.suchymips.de
*
/* ***** RBarc/QR-Code 1 *****
/* ***** definition of encoding data and data types *****
/: DEFINE &QVRAR1% = '(>'
/: DEFINE &DFLAG1% = 'T'
/: DEFINE &QVRAR2% = 'RS'
/: DEFINE &DFLAG2% = 'F'
/: DEFINE &QVRAR3% = '06'
/: DEFINE &DFLAG3% = 'T'
/: DEFINE &QVRAR4% = 'GS'
/: DEFINE &DFLAG4% = 'F'
/: DEFINE &QVRAR5% = '2L'
/: DEFINE &DFLAG5% = 'T'
/*
/* ***** RBarc/QR-Code 1 *****
/* ***** definition of optional barcode parameters *****
/: DEFINE &RES% = 150
/: DEFINE &QVERSION% = 0
/: DEFINE &ECC_LEVEL% = 'L'
/: DEFINE &X_DIM% = 4
/: DEFINE &XPOS% = '50.00'
/: DEFINE &YPOS% = '0.00'
/: DEFINE &AUTOHEIGHT% = 'N'
/*
/* ***** RBarc/QR-Code 1 *****
/* ***** performing barcode generation *****
/: PERFORM GEN_QRCODE_SS IN PROGRAM Z_SET_QRCODE
/: USING &QVRAR1%
/: USING &DFLAG1%
/: USING &QVRAR2%
/: USING &DFLAG2%
/: USING &QVRAR3%
/: USING &DFLAG3%
/: USING &QVRAR4%
/: USING &DFLAG4%
/: USING &QVRAR5%
/: USING &DFLAG5%
/: USING &RES%
/: USING &QVERSION%
/: USING &ECC_LEVEL%
/: USING &X_DIM%
/: USING &XPOS%
/: USING &YPOS%
/: USING &AUTOHEIGHT%
/: CHANGING &QRNAME%
/: CHANGING &QRWIDTH%
/: CHANGING &RESULT%
/: ENDPERFORM
/*
/* ***** RBarc/QR-Code 1 *****
/* ***** including barcode bitmap *****
/: BITMAP &QRNAME% OBJECT GRAPHICS ID BMAP TYPE BMON XPOS &XPOS% MM
/*
/* ***** RBarc/QR-Code 1 *****
/* ***** deleting barcode bitmap *****
/: PERFORM DEL_QR_SS IN PROGRAM Z_SET_QRCODE
/: USING &QRNAME%
/: CHANGING &RESULT%
/: ENDPERFORM
/* ***** END of QR-Code 1 *****
```

Explanations:

**DEFINE &DMVAR1& = ['(>Ä'
DEFINE &DFLAG1& = 'T'**

...

Definition of variables for encoding. There is no limitation in the number of variables. The names of variables are preset and can not be changed. The data to be encrypted must be stored in **DMVARx**, where **x** is the number of variable starting with **1** in ascending order.

DEFINE &RES& = 150

Definition of the bitmap resolution, as described in 6.3. If the parameter is omitted, then the corresponding **USING** variable in the **PERFORM GEN_DATAMATRIX_SS** command must be removed.

DEFINE &MSIZE& = 0

Definition of the matrix size, as described in 6.3. If the parameter is omitted, then the corresponding **USING** variable in the **USING PERFORM GEN_DATAMATRIX_SS** command must be removed.

DEFINE &X_DIM& = 4

Definition of the horizontal size of the smallest module as described under 6.3. If the parameter is omitted, then the corresponding **USING** variable in the **PERFORM GEN_DATAMATRIX_SS** command must be removed

DEFINE &XPOS& = '50.00'

Definition of the horizontal position of the bar code graphic relatively to the left window border. The unit is millimeters. The value must be single quoted and be heigher than or equal zero. If the parameter is omitted, then the corresponding **USING** variable in the **PERFORM GEN_DATAMATRIX_SS** command must be removed.

DEFINE &YPOS& = '0.00'

Definition of the vertical position of the bar code graphic, relatively to the recent line. The unit is millimeters. The value must be single quoted and be heigher than or equal zero. If the parameter is omitted, then the corresponding **USING** variable in the **PERFORM GEN_DATAMATRIX_SS** command must be removed.

DEFINE &AUTOHEIGHT& = 0

Definition of the bar code space treatment, like described in 6.3. If the parameter is omitted, then the corresponding **USING** variable in the **PERFORM GEN_DATAMATRIX_SS** command must be removed.


```

PERFORM GEN_DATAMATRIX_SS IN PROGRAM Z_SET_DATAMATRIX
USING &DMVAR1&
USING &DFLAG1&
USING &DMVAR2&
USING &DFLAG2&
USING &DMVAR3&
USING &DFLAG3&
USING &DMVAR4&
USING &DFLAG4&
USING &DMVAR5&
USING &DFLAG5&
USING &RES&
USING &MSIZE&
USING &X_DIM&
CHANGING &DMNAME&
CHANGING &DMWIDTH&
CHANGING &RESULT&
ENDPERFORM

```

Passing the encoding variables "**DMVAR...**" with their variable category "**DFLAG...**" and optional bar code parameters "**RES**" (Resolution), "**MSIZE**" (Matrix Size) and "**X_DIM**" (dimension of the smallest module) to the form routine **GEN_DATAMATRIX_SS** in the program **Z_SET_DATAMATRIX**. The number of variables is not limited and can be extended taking into account the rule of variable numbering as described above. The program returns the name of the generated bar code (parameter "**DMNAME**"), the bar code width in dots of recent resolution ("**DMWIDTH**") and the result ("**RESULT**"). If no error occurs, then **RESULT = 'No errors'**.

BITMAP &DMNAME& OBJECT GRAPHICS ID BMAP TYPE BMON XPOS &XPOS& MM

The bar code will be embedded into the form dynamically. The parameter "**XPOS**" moves the bar code horizontally inside the window in which it was embedded.

```

PERFORM DEL_DM_SS IN PROGRAM Z_SET_DATAMATRIX
USING &DMNAME&
CHANGING &RESULT&
ENDPERFORM

```

After including the bar code into the form, it will be deleted from the system. The parameter **RESULT** returns the result of the operation. If no errors occurs, then **RESULT = 'No errors'**.

It is very important to delete the bar. Otherwise, it remains in the system must be maintained and subsequently deleted. Check with the transaction "**SE78**" whether graphics that begin with "ZDM" and generated during testing are still in the system. Graphics can be deleted with the program "**ZDELBMP**", which can be found in the "**UTILITY**" subdirectory.

Remember:

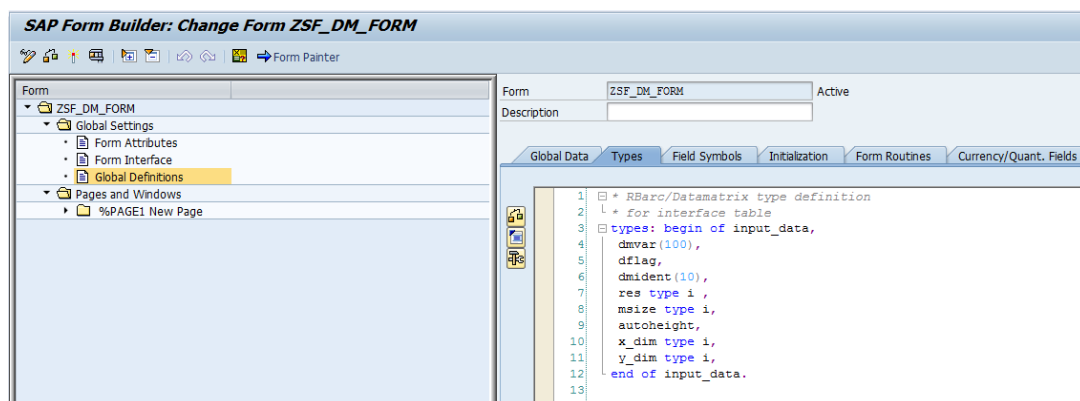
- The number of encoding variables (**DMVAR...**) is not limited.
- The corresponding variable type (**DMFLAG...**) must be assigned to each variable.
- The encoding variables must be numbered without gaps, starting with **1**.
- The type **DFLAGx** must have the same number as the corresponding variable **DMVARx**.
- Each variable **DMVARx** and the corresponding type **DFLAGx** must follow each other as **USING** Parameter.
- Only actually present or explicitly with the **DEFINE** declared variables may be passed as **USING** parameters.
- The parameters "**RES**", "**MSIZE**", "**X_DIM**", "**XPOS**", "**YPOS**" and **AUTOHEIGHT** are optional. If not used, then default values are used automatically.

6.5 Datamatrix printing with Smart Forms

6.5.1 Bar code definition in a Smart Forms form

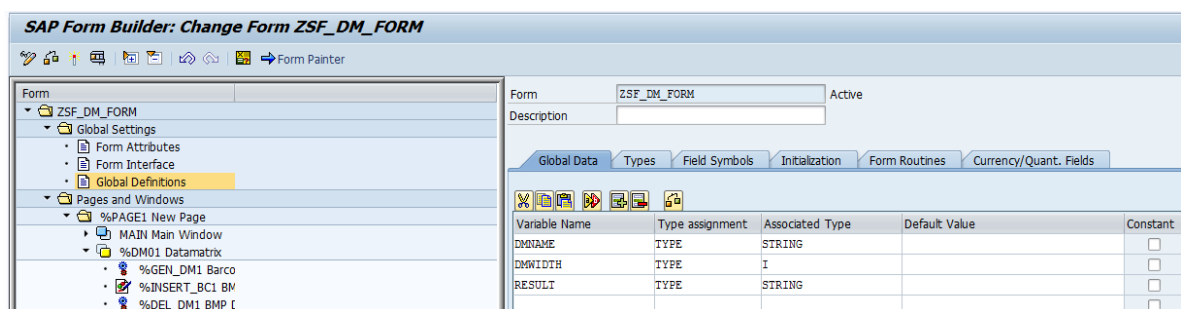
1. Define under tabpaint "Types" in the tree node "Global Definitions" the type input_data:

```
types: begin of input_data,  
  dmvar(100),  
  dflag,  
  dmident(10),  
  res type i ,  
  msize type i,  
  autoheight,  
  x_dim type i,  
  y_dim type i,  
end of input_data.
```

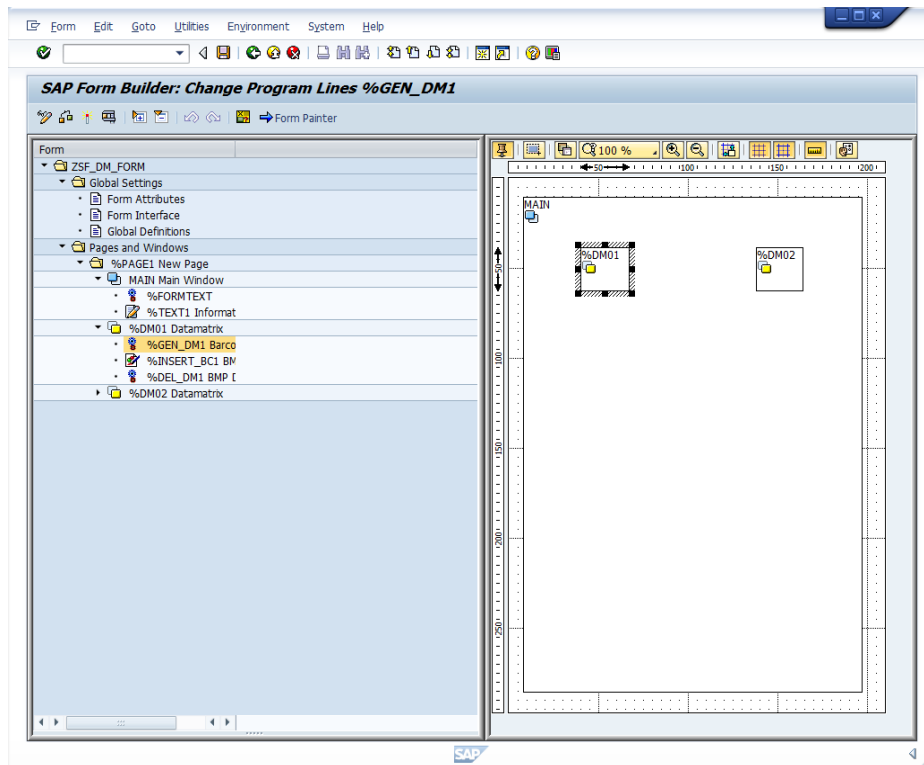


2. Define in the tabpaint "Global Data" from the tree node "Global Definitions" the following variables:

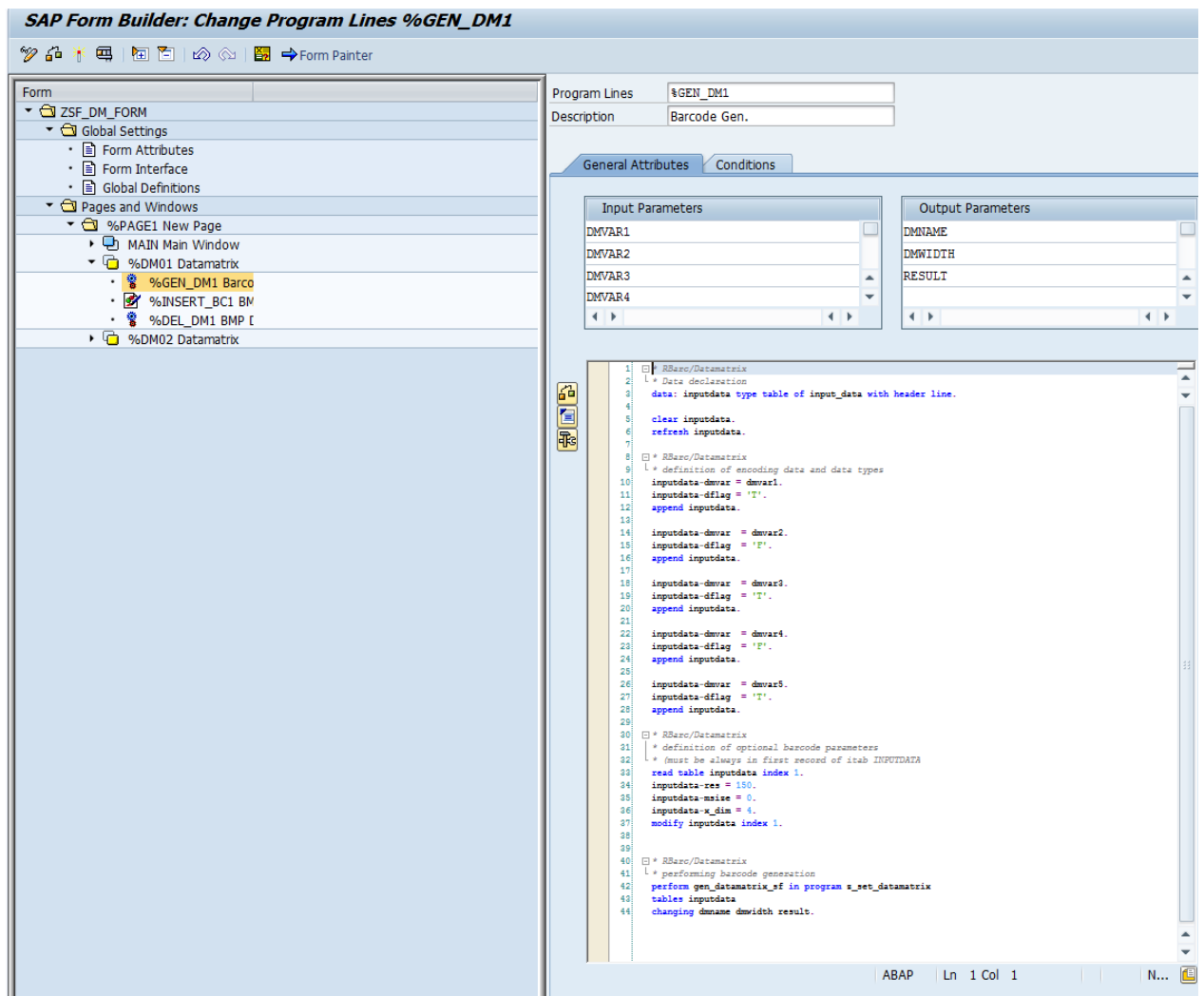
DMNAME	Type	String
DMWIDTH	Type	I
RESULT	Type	String



3. Create a window, which is large enough to contain the bar code label.
This is the windows **"DM01 Datamatrix"** in the example below.



4. Create in the new window a nod of type "Program Lines".



- Enter all necessary encoding variables as **Input Parameters**.
- Enter the variables **DMNAME**, **DMWIDTH** and **RESULT** as **Output Parameter**. **DMWIDTH** and **RESULT** is for your own use.
- Declare the **internal Table INPUTDATA**
`data: inputdata type table of input_data with header line.`
- Delete the content of **INPUTDATA** (important, for example if the form has several pages where the same routine is used).
`clear inputdata.`
`refresh inputdata.`

- Write to each encoding variable into the table field **DMVAR** and the corresponding data type in the table field **DFLAG** and append each record to the table **INPUTDATA**:

```
inputdata-dmvar = dmvar1.  
inputdata-dflag = 'T'.  
append inputdata.
```

... and so on for each encoding variable.

The number of encodable variables is unlimited. The variables with their corresponding types must be contiguously append to the table. Thus, each variable with its corresponding data type is a data set (= record) in the table.

- After all variables to be encoded were append to the internal table **INPUTDATA**, read the first record of this table and enter there the desired values for **RES** (resolution), **MSIZE** (matrix size) and **X_DIM** (size of the smallest module) and modify the record accordingly

```
read table inputdata index 1.  
Inputdata-res = 150.  
inputdata-msize = 0.  
inputdata-x_dim = 4.  
modify inputdata index 1.
```

This part is optional. If it is not used, then default values are used for the parameters **RES**, **MSIZE** and **X_DIM**.

- Pass the table **INPUTDATA** to the form routine **GEN_DATAMATRIX_SF** in the program **Z_SET_DATAMATRIX**

```
perform gen_datamatrix_sf in program z_set_datamatrix  
tables inputdata  
changing dmname dmwidth result.
```

- The program returns the name of the generated bar code (parameter "**DMNAME**"), the bar code width in dots of recent resolution ("**DMWIDTH**") and the result ("**RESULT**"). If no error occurs, then **RESULT = 'No errors'**.

5. Create (still in the same window) a second node of type **"Graphic"** and fill the fields **"Name"**, **"Object"**, and **"ID"** like presented in the example below:

The screenshot shows the SAP Form Builder interface with the title "SAP Form Builder: Change Graphic %INSERT_BC1". The left pane displays a tree structure of the form, with the node "%INSERT_BC1 BMP" selected. The right pane shows the configuration for this graphic node. The "Graphic" field is set to "%INSERT_BC1" and the "Description" is "BMP Instert". The "General Attributes" tab is active, showing the "Name" as "&DMNAME&", "Object" as "GRAPHICS", and "ID" as "BMAP". The "Output Options" section has three radio buttons: "Black and White Bitmap Image (BMON)" (selected), "Color Bitmap Image (BCOL)", and "Determine Dynamically (BMON, BCOL)". The "Technical Attributes" section shows "Resolution" set to "DPI".

Name = &DMNAME&

Object = GRAPHICS

ID = BMAP

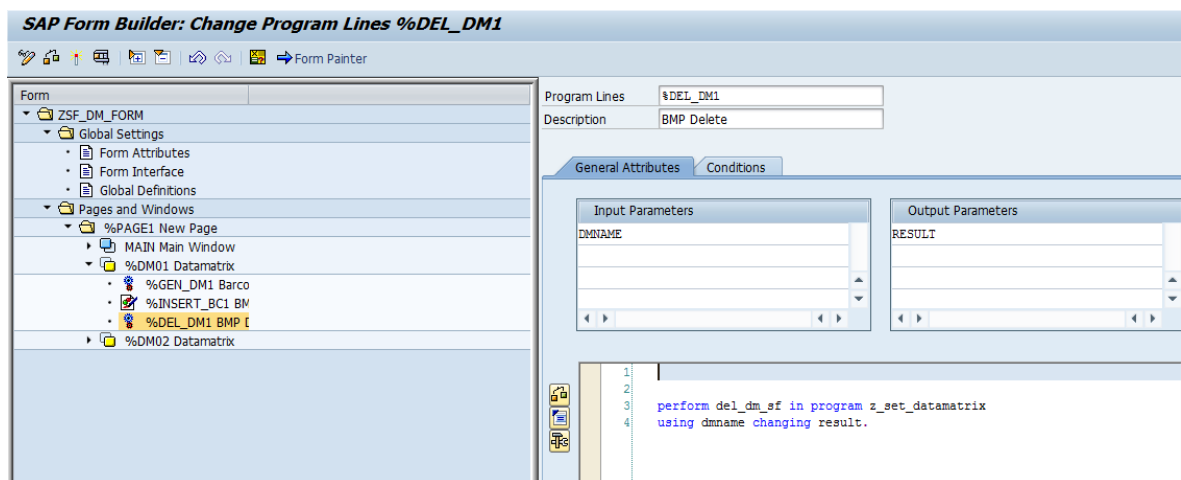
Resolution = leave it empty

6. Create (still in the same window) a third node of type "**Program Lines**" and insert following program lines:

```
perform del_dm_sf in program z_set_datamatrix  
using dmname  
changing result.
```

Enter as **Input Parameter DMNAME**.

Enter as **Output Parameter RESULT** (optionally, if you want to use it).



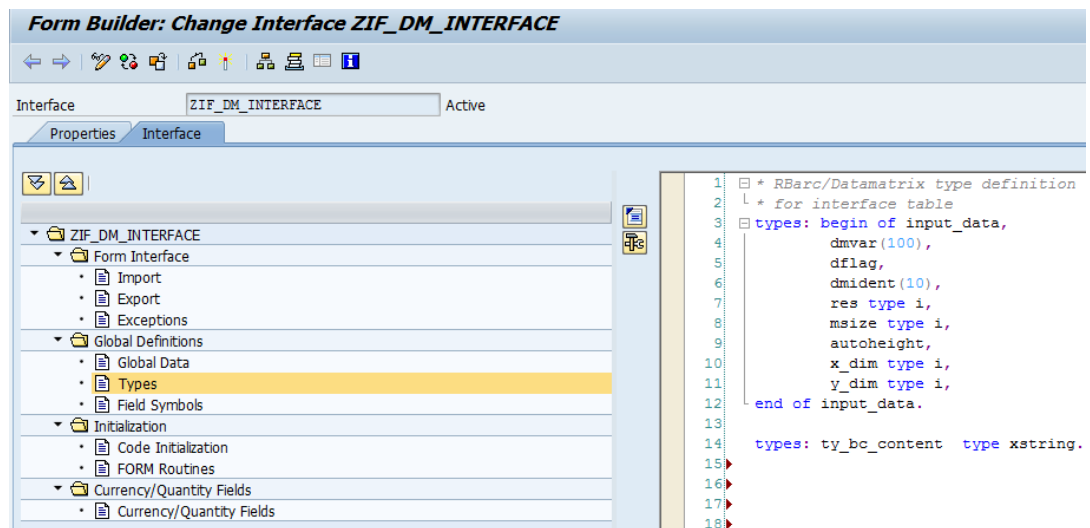
6.6 Datamatrix printing with Interactive Forms

6.6.1 Enhancement of the Interactive Forms Interface

1. Define the types `input_data` and `ty_bc_content` in the node „Global Definitions→ Types“ as follows:

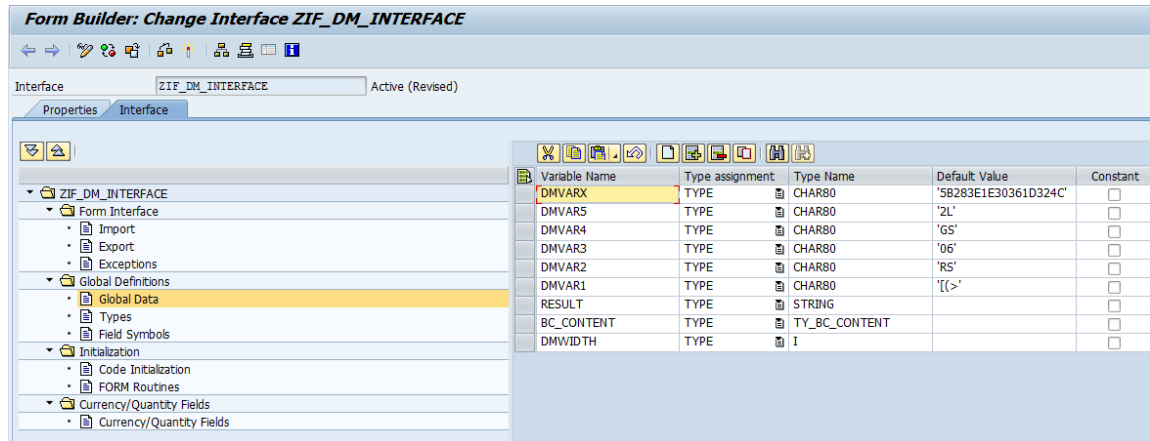
```
types: begin of input_data,  
    dmvar (100) ,  
    dflag,  
    dmident (10) ,  
    res type i,  
    msize type i,  
    autoheight,  
    x_dim type i,  
    y_dim type i,  
end of input_data.
```

```
types: ty_bc_content type xstring.
```



- Define following global variables in the node „**Global Definitions**→ **Global Data**“.

DMWIDTH	TYPE	I
RESULT	TYPE	STRING
BC_CONTENT	TYPE	TY_BC_CONTENT



Notice:

In the example above the encoding variables **DMVAR1** to **DMVAR5** were also defined as global variables. In general, however, the variables will come from the workflow of data processing. You then need to ensure by appropriate means that the required variables are also visible in the interface. Usual, it is then not necessary to define such variables as global variables, as in the case of the field **MATNR** from the table **MARD**.

For multiple bar codes in a form more global variable of Type **TY_BC_CONTENT** can be defined.

3. Insert following program lines into the node „Global Definition→ Code Initialization“ (here is an example with 5 encoding variables).

data: inputdata type table of input_data with header line.

clear inputdata.
refresh inputdata.

inputdata-dmvar = dmvar1.
inputdata-dflag = 'T'.
append inputdata.

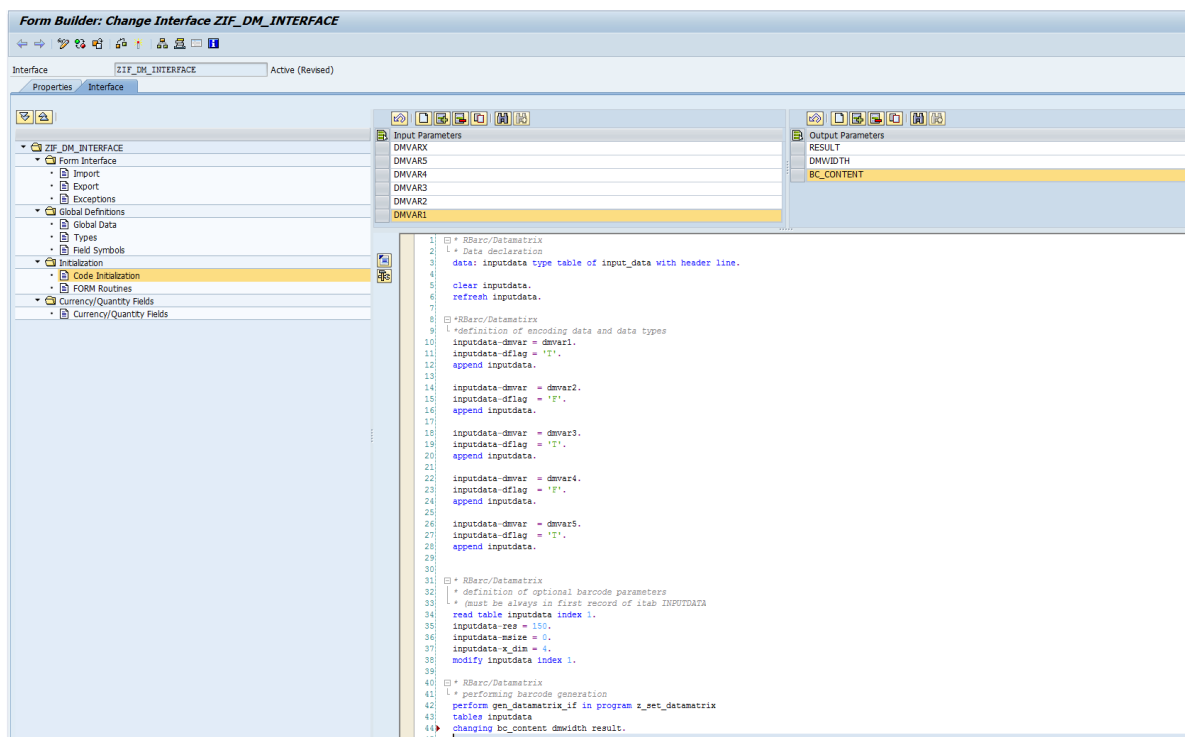
inputdata-dmvar = dmvar2.
inputdata-dflag = 'F'.
append inputdata.

inputdata-dmvar = dmvar3.
inputdata-dflag = 'T'.
append inputdata.

inputdata-dmvar = dmvar4.
inputdata-dflag = 'F'.
append inputdata.

inputdata-dmvar = dmvar5.
inputdata-dflag = 'T'.
append inputdata.

read table inputdata index 1.
inputdata-res = 150.
inputdata-msize = 0.
inputdata-x_dim = 4.
modify inputdata index 1.



Explanations:

- Enter as **Input Parameter** all encoding variables.
- Enter as **Output Parameter** the variables **BC_CONTENT** and **RESULT**.
- Declare the internal table **INPUTDATA**
`data: inputdata type table of input_data with header line.`
- Delete the content of **INPUTDATA** (important, for example if the form has several pages where the same routine is used).
`clear inputdata.`
`refresh inputdata.`
- Write each encoding variable into the table field **DMVAR** and the corresponding data type into the table field **DFLAG** and append each record to the table **INPUTDATA**:
`inputdata-dmvar = dmvar1.`
`inputdata-dflag = 'T'.`
`append inputdata.`

... and so on for each encoding variable.

The number of encodable variables is unlimited. The variables with their corresponding types must be contiguously written into the table. Thus, each variable with its corresponding data type is a data set (= record) in the table.
- After all variables to be encoded were append to the internal table **INPUTDATA**, read the first record of this table and enter there the desired values for **RES** (resolution), **MSIZE** (matrix size) **ECC_LEVEL** (Error Correction Level) **X_DIM** (size of the smallest module) and modify the record accordingly
`read table inputdata index 1.`
`inputdata-res = 150.`
`inputdata-dmsize = 0.`
`inputdata-x_dim = 4.`
`modify inputdata index 1.`

This part is optional. If it is not used, then default values are used for the parameters **RES**, **MSIZE**, and **X_DIM**.
- Pass the table **INPUTDATA** to the form routine **GEN_DATAMATRIX_IF** in the program **Z_SET_DATAMATRIX**
`perform gen_datamatrix_if in program z_set_datamatrix`
`tables inputdata`
`changing bc_content dmwidth result.`

Return paramaters are: **bc_content** (this is he bar code graphic), **dmwidth** (the bar code bitmap width in dots oft he current resolution) und **result** (the result). If the bar code generation was proceeded without errors, the result contains the message: **result = 'No errors'**.

Important:

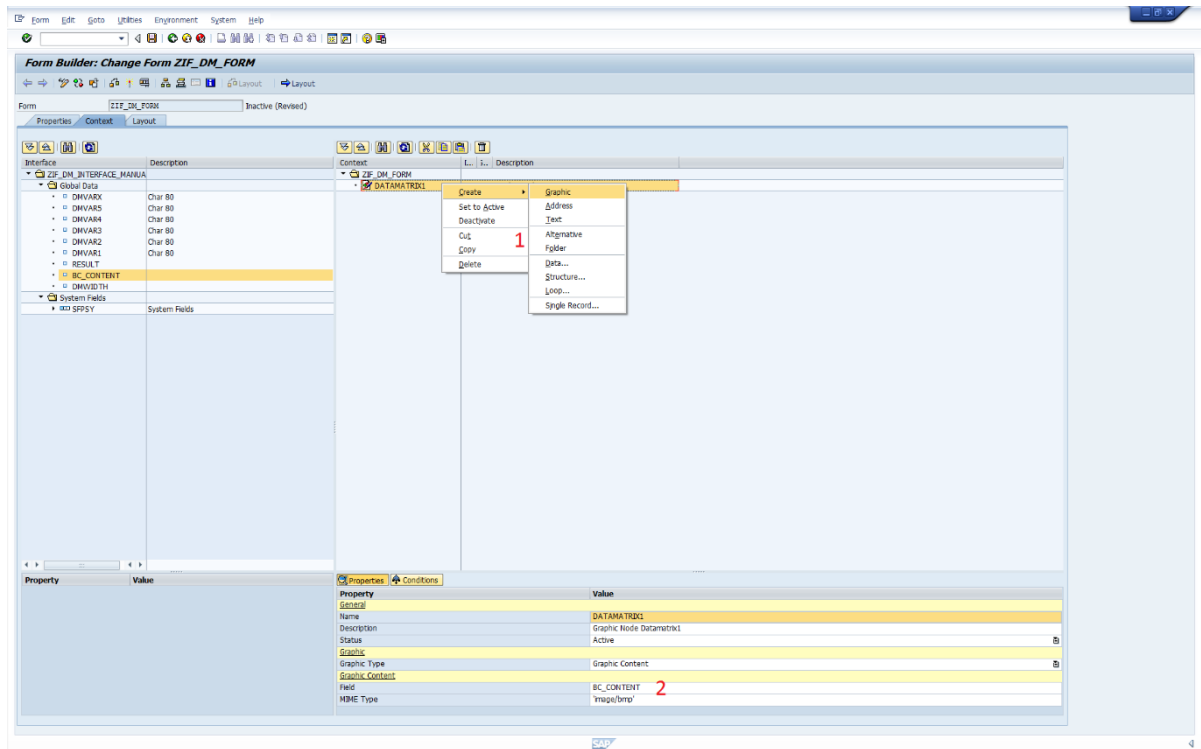
Please remember that all used encoding variables have to be also advertised in the node „Initialization“, meaning you have to define them as input parameters together with other parameters..

6.6.2 Enhancement of the Interactive Forms Form

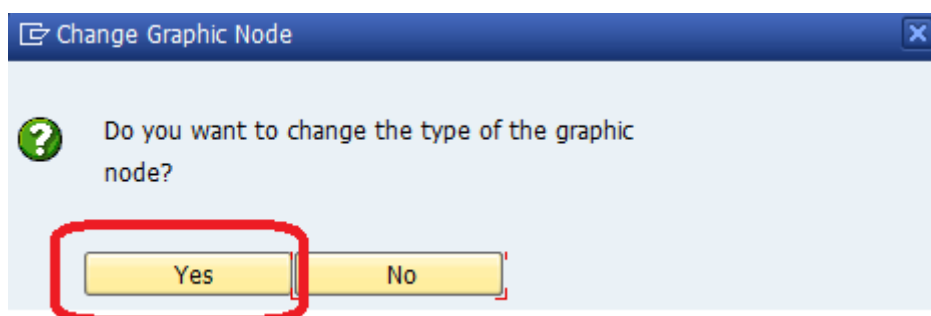
In the **Interactive Forms** form the bar code created with **RBarc/DataMatrix** is displayed as an image object. Therefore, at first the context has to be expanded by the necessary elements for a bar code.

1. Select in the form (Transaktion **SFP**) in the context menu (right windows side) the form name, click on the right mouse button and select from context menu **Create→Graphik** in order to create a graphic object.

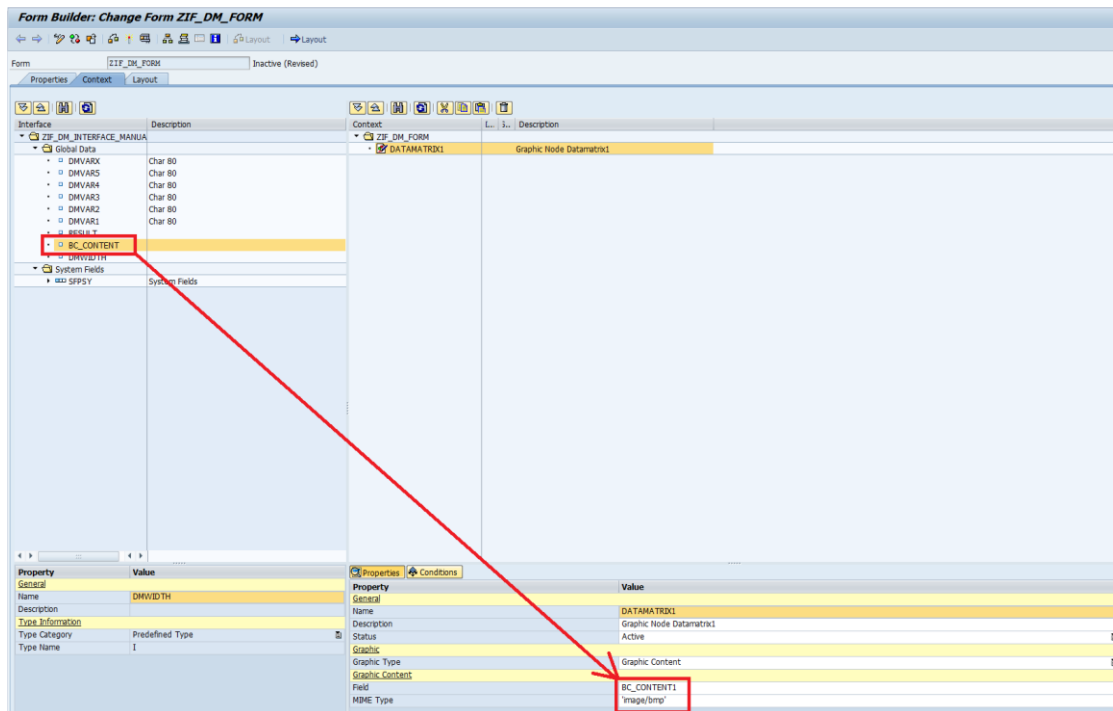
Assign a name and description to the created graphic object. In the example below it is **DATAMATRIX1**.



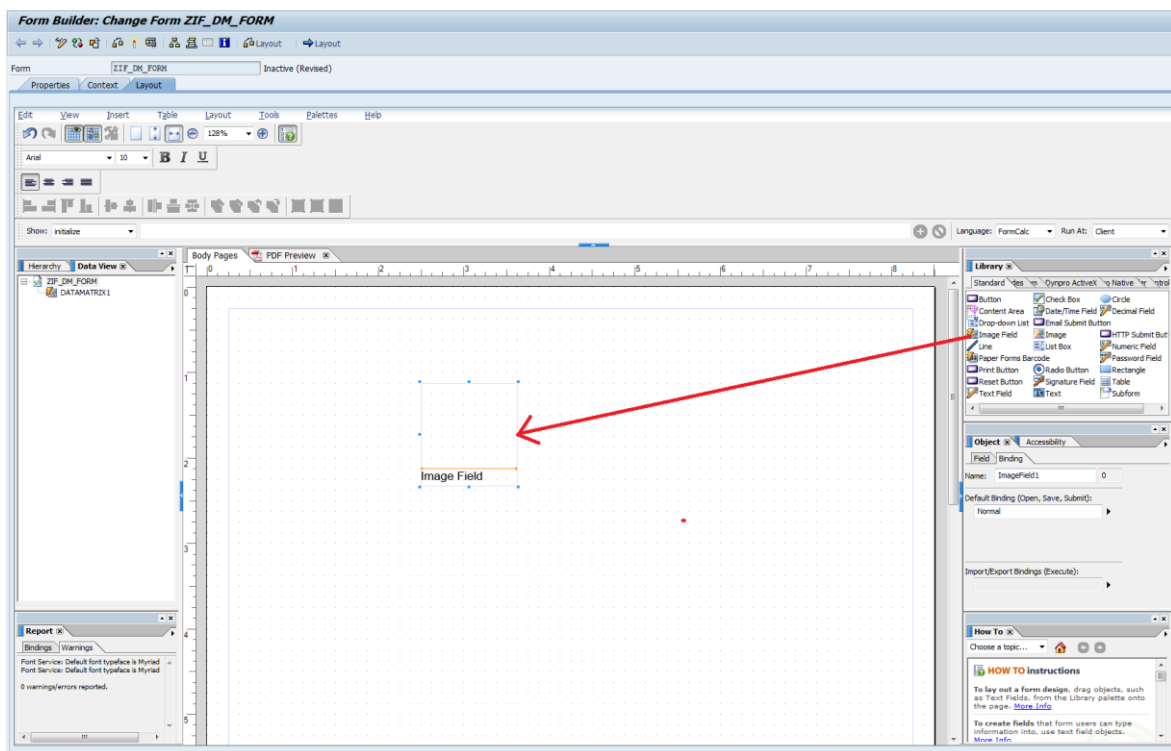
2. Change the graphic type to „**graphic content**“ (see point 2 in the screenshot above) and confirm the popup with „Yes“.



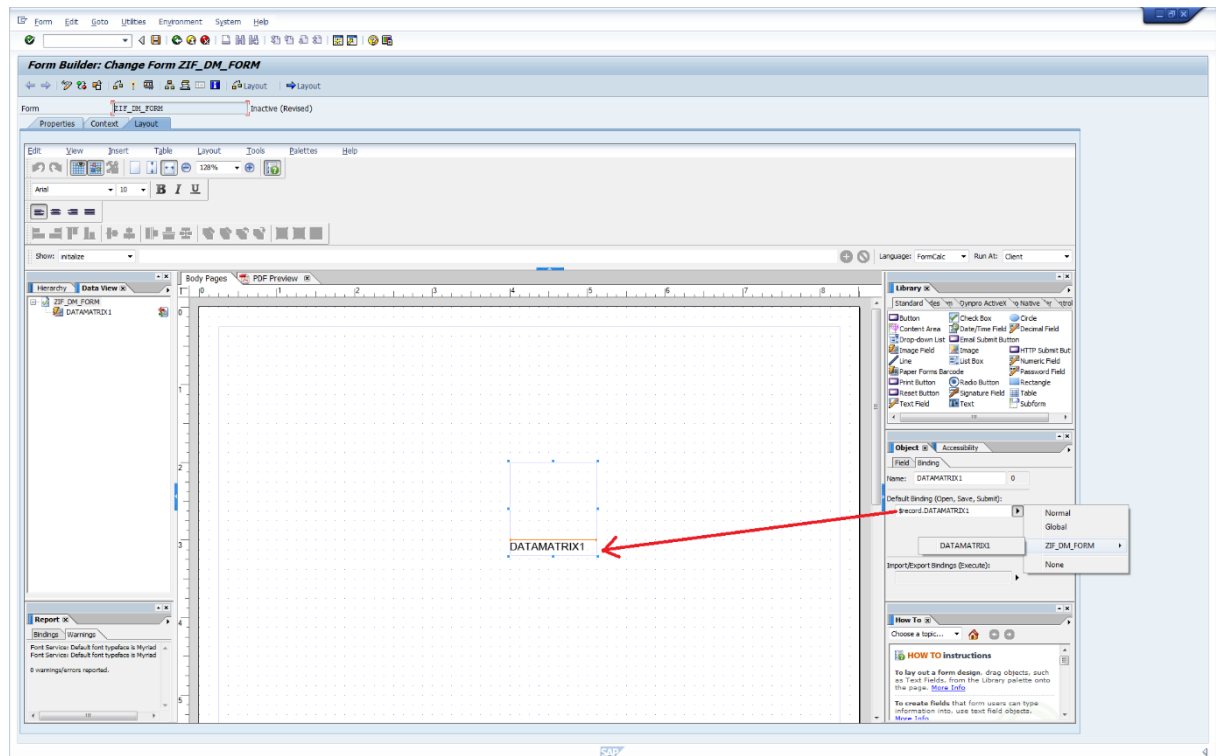
- Now, assign tie the graphic **DATAMATRIX1** to the data field containing the bar code content, by drawing, with the left mouse button pressed, the field **BC_CONTENT** from the interface to the appropriate field on the right. Provide it with ,Image/bmp' as a **MIME-Type**.



- Change to the layout view of the Form Builders and create an element of the type „image field“.



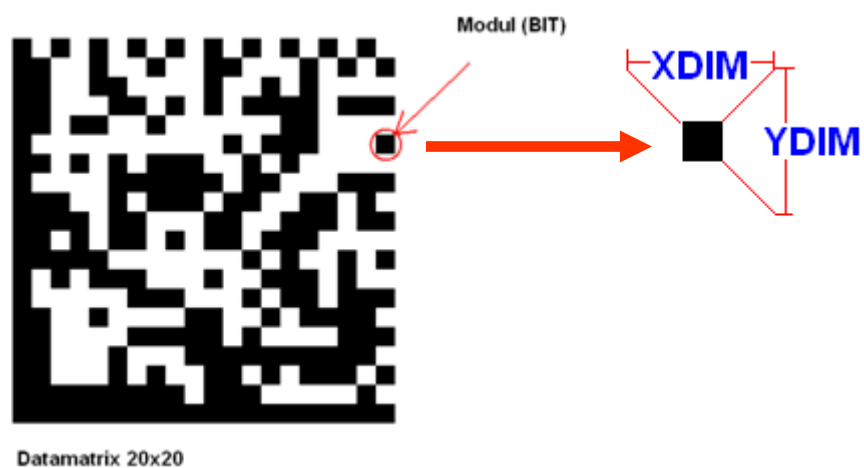
5. As a binding to the image field select the graphic **DAMATRIX1** previously defined in the context.



Ready!

The newly generated binding of the image field to the graphic **DATAMATRIX1** will cause, that during the run time the Datamatrix bar code generated by **RBarc/Datamatrix** will be inserted ON-THE-FLY into the form at the selected place.

6.7 Definition of bar code properties.



A DATAMATRIX bar code consists of black and white modules whose size is variable.










Following matrix size for a datamatrix label may be defined:

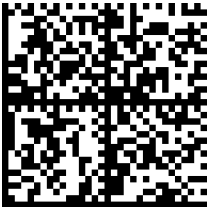
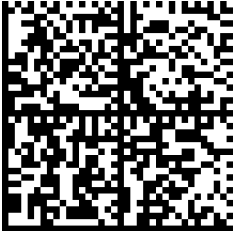
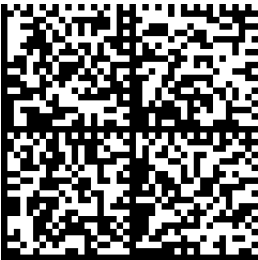
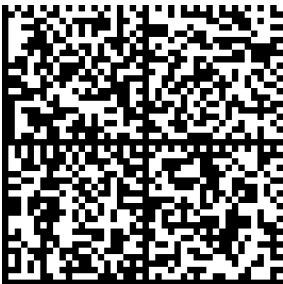

Msize	Number of Modules	Capacity of data codewords
0	auto	Up to the quantity of data
1	10x10	3
2	12x12	5
3	14x14	8
4	16x16	12
5	18x18	18
6	20x20	22
7	22x22	30
8	24x24	36
9	26x26	44
10	32x32	62
11	36x36	86
12	40x40	114
13	44x44	144
14	48x48	174
15	52x52	204
16	64x64	280
17	72x72	368
18	80x80	456
19	88x88	576
20	96x96	696
21	104x104	816
22	120x120	1050
23	132x132	1304
24	144x144	1558
25	8x18	5
26	8x32	10
27	12x26	16
28	12x36	22
29	16x36	32
30	16x48	49



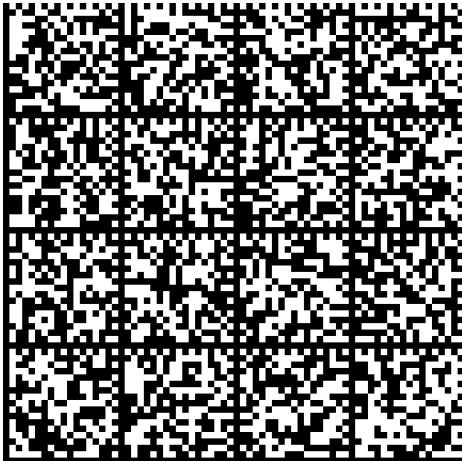
The table above shows the data codeword capacity of a Datamatrix label depending on the selected matrix size (parameter **msize**). Note, that this value does not reflect the capacity of real data to be encoded. Datamatrix uses additional codewords to change automatically the encoding scheme, so that sometimes the capacity of real data is higher than the capacity of codewords, but sometimes it also could be lower. For more information refer to the official specification of the Datamatrix bar code available at AIM.


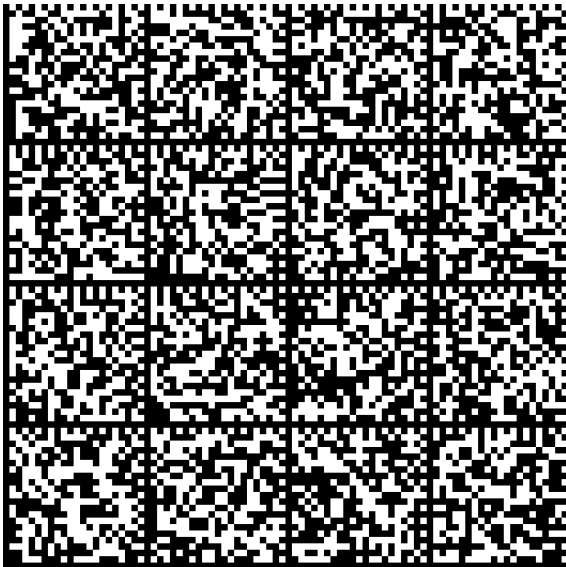
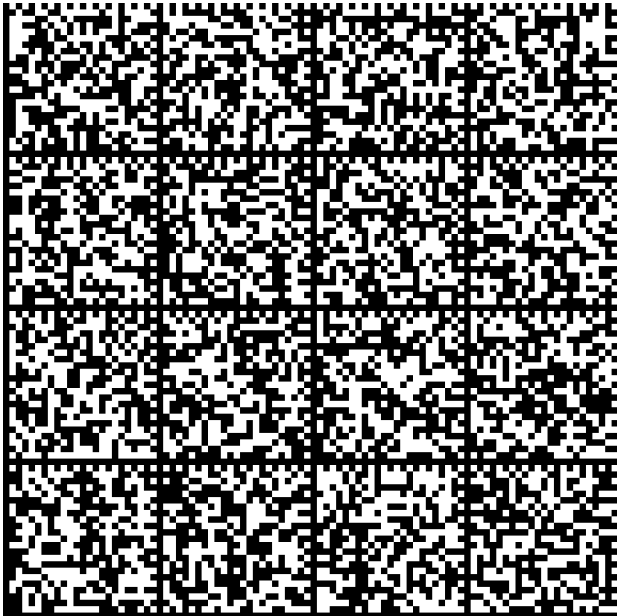
Notice:

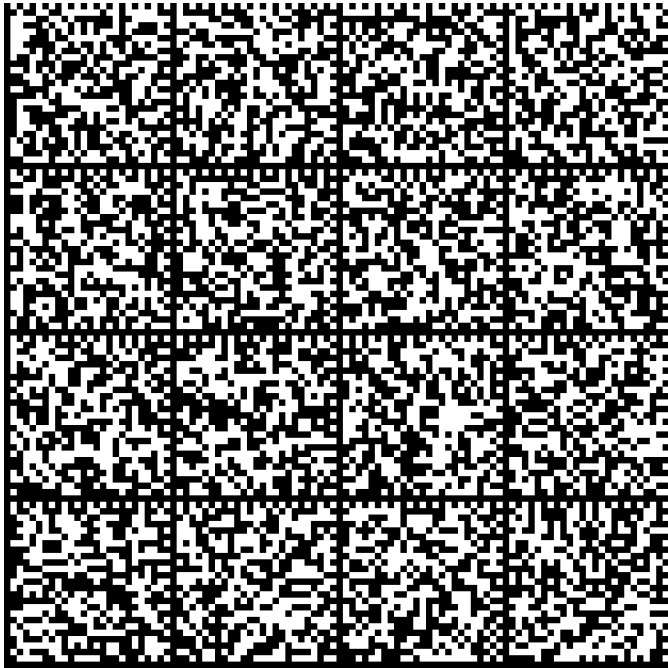
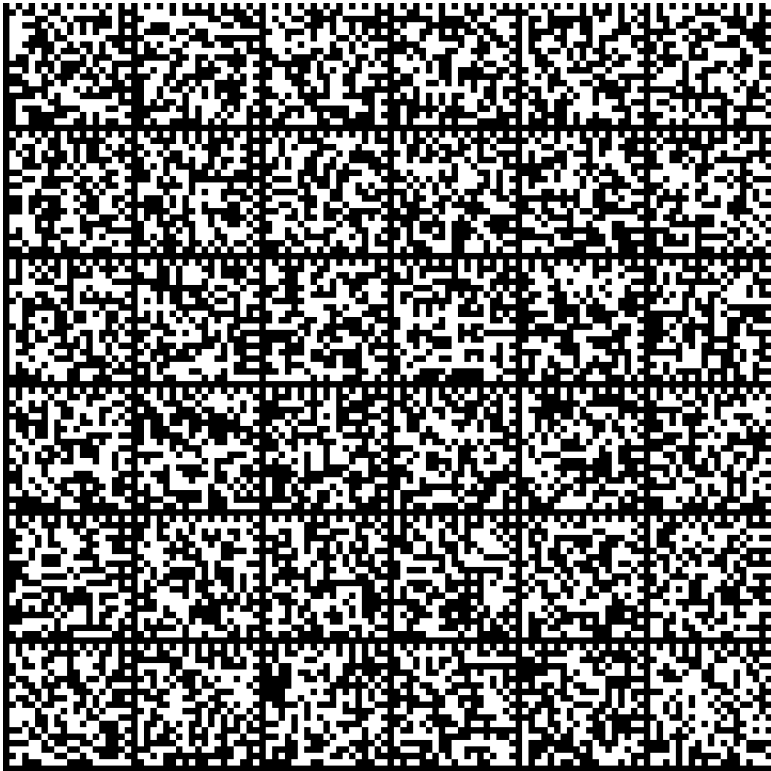
msize = 0 causes, that the matrix size will be calculated automatically with the smallest possible value. The advantage here is, that the programmer doesn't have to care about the data size send to **RBarc/Datamatrix**. The disadvantage is, that the labels on different pages may be of different size, due to quantity of data, which was encoded. If a fix bar code size is required, the programmer should calculate the maximum possible data size and then to define a bar code label of a fix size (e.g. **msize = 5**). The disadvantage of this procedure is, that if the amount of calculated data codewords will exceed the defined bar code capacity, an error will occur.

Matrix Size	Bar Code Example
10x10	
12x12	
14x14	
16x16	
18x18	
20x20	
22x22	
24x24	
26x26	

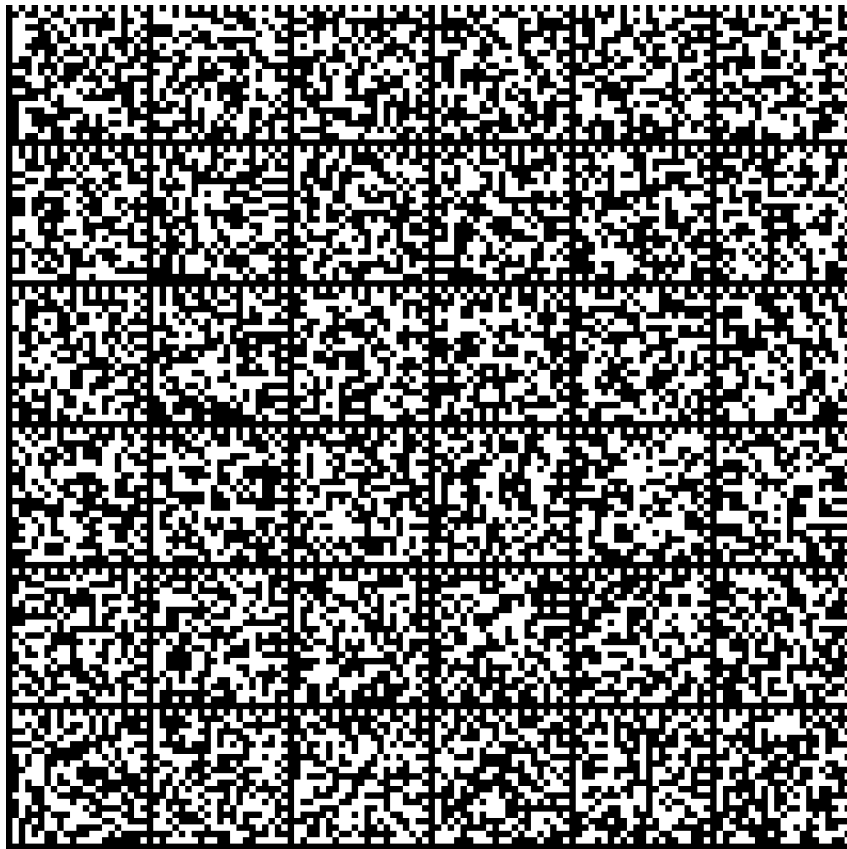
32x32	
36x36	
40x40	
44x44	
48x48	

52x52	
64x64	
72x72	

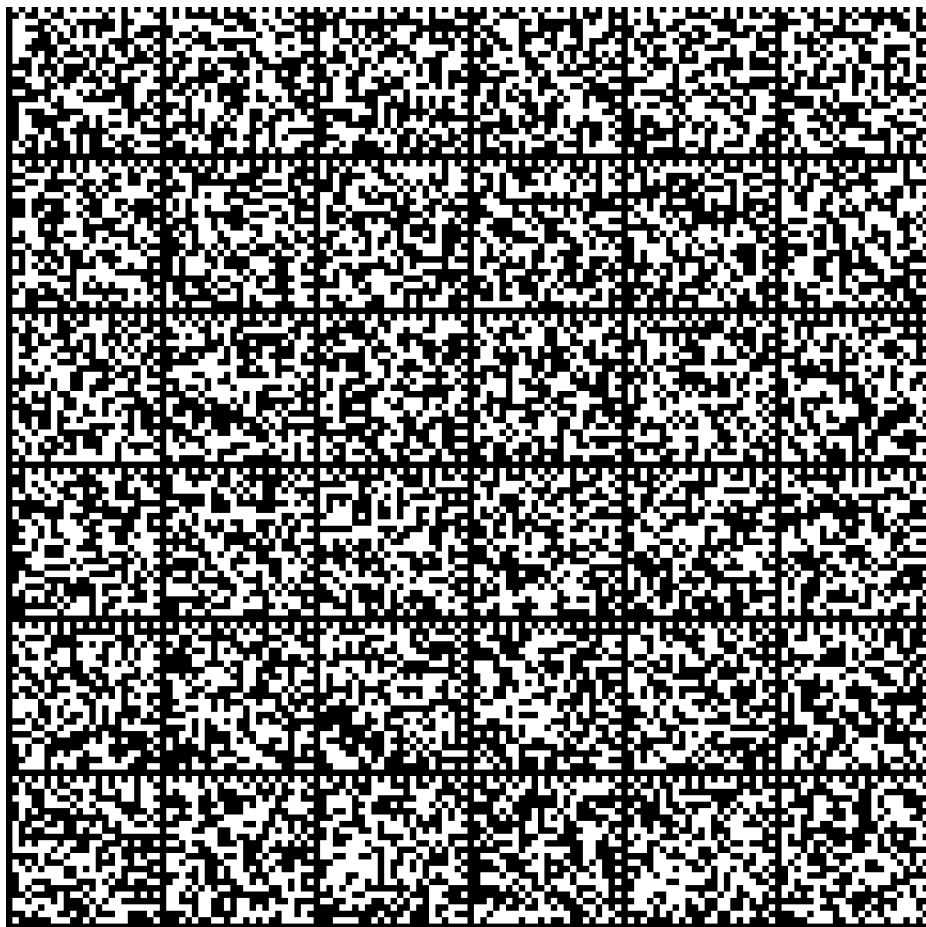
80x80	
88x88	
96x96	




104x104	
120x120	

132x132



144x144



8x18	
8x32	
12x26	
12x36	
16x36	
16x48	

Example of a job definition for a SAPscript Form.

6.8 Job

The following fields shall be encoded in a Datamatrix bar code label:

VBELN from the table **LIKP**,
VBELN from the table **VBRK** und
ADDRNUMBER from the table **ADRT**.

The variables shall be separated by a **GS** (Group Separator). The bar code should be printed **50 Mm** right from the left border of the window, it was defined in. The module size shall be **1 Mm** wide and **1 Mm** heigh. The size of Matrix shall be calculated automatically

Solution:

1. Define the 3 variables for encoding, their type (dflag) and variables with bar code properties (**RES**, **MSIZE** and **X_DIM**) Pass all values in one step to the form routine **GEN_DATAMATRIX_SS** in the report **Z_SET_DATAMATRIX**. Then define an include command line for the generated bar code and finally include a command to delete the bar code from the system.

```
/: DEFINE &DMVAR1& = &LIKP-VBELN&  
/: DEFINE &DFLAG1& = 'T'  
/: DEFINE &DMVAR2& = 'GS'  
/: DEFINE &DFLAG2& = 'F'  
/: DEFINE &DMVAR3& = &VBRK-VBELN&  
/: DEFINE &DFLAG3& = 'T'  
/: DEFINE &DMVAR4& = 'GS'  
/: DEFINE &DFLAG4& = 'F'  
/: DEFINE &DMVAR5& = &ADRT-ADDRNUMBER&  
/: DEFINE &DFLAG5& = 'T'  
/: DEFINE &RES& = 150  
/: DEFINE &MSIZE& = 0  
/: DEFINE &X_DIM& = 6
```

(goto next page)

```
/: PERFORM GEN_DATAMATRIX_SS IN PROGRAM Z_SET_DATAMATRIX
```

```
/: USING &DMVAR1&  
/: USING &DFLAG1&  
/: USING &DMVAR2&  
/: USING &DFLAG2&  
/: USING &DMVAR3&  
/: USING &DFLAG3&  
/: USING &DMVAR4&  
/: USING &DFLAG4&  
/: USING &DMVAR5&  
/: USING &DFLAG5&  
/: USING &RES&  
/: USING &MSIZE&  
/: USING &X_DIM&  
/: CHANGING &DMNAME&  
/: CHANGING &DMWIDTH&  
/: CHANGING &RESULT&  
/: ENDPERFORM
```

```
/: BITMAP &DMNAME& OBJECT GRAPHICS ID BMAP TYPE BMON XPOS &XPOS& MM
```

```
/: PERFORM DEL_DM_SS IN PROGRAM Z_SET_DATAMATRIX
```

```
/: USING &DMNAME&  
/: CHANGING &RESULT&  
/: ENDPERFORM
```

In the implementation shown above, initially a type is assigned to each variable (eg. **&LIKP-VBELN&** was defined as text). The separator "**GS**" was designated as a function code with **DFLAG = 'F'**.

In the next step bar code properties **RES**, **MSIZE** and **X_DIM** were defined.

All defined variables and parameters were passed in one step to the form routine in **GEN_DATAMATRIX_SS** in the program **Z_SET_DATAMATRIX**

The graphic generated by **RBarc / Damatrix** with the name **&DMNAME&** is included in the form with the **BITMAP...** statement.

Finally, the image is deleted from the system by calling the form routine **DEL_DM_SS** in program **Z_SET_DATAMATRIX**.

7 Example of a job definition for a Smart Forms Form.

7.1 Job.

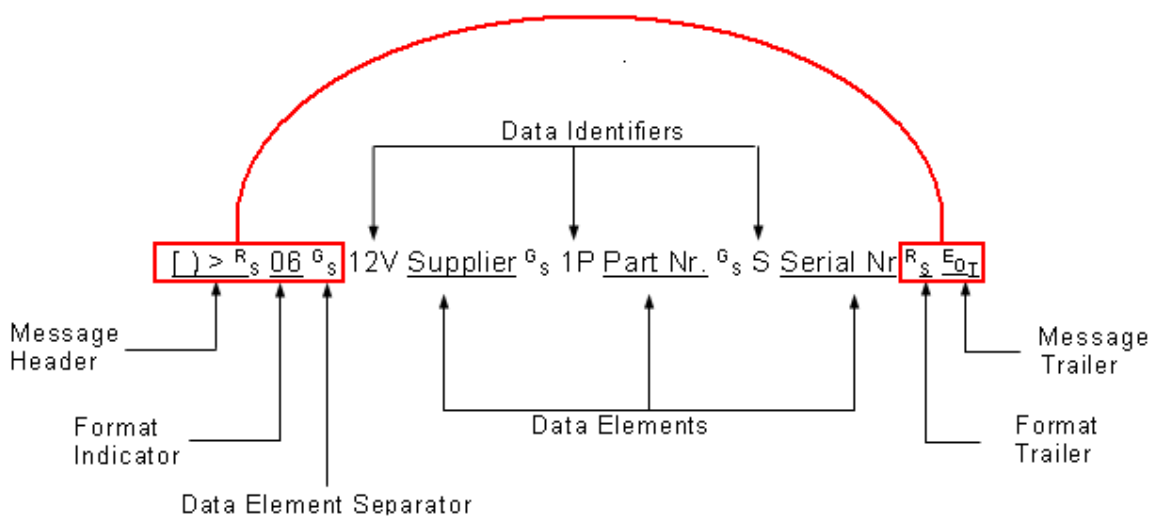
3 variables shall be encoded:

The supplier number "**SupplierNo**",

The part number "**PartNo**"

The serial number of the part "**SerNo**".

All variables shall be encapsulated in the **Message Envelop** (see picture below). The module shall be **1 Mm** wide and **1 Mm** high. The matrix size shall be calculated automatically.



The "**Message Envelop**" is a standardized way of data transfer to the ANSI standard MH10.8.7. The encoding data will be preceded by a message header – up to the standard " $[] >^R_s 06^G_s$ " or " $[] >^R_s 05^G_s$ " end finished by the Trailer $^R_s E_{OT}$. Each variable will be preceded by an indicator (eg. 12V for Supplier) and all variables will be separated by the fields separator G_s .

Solution:

1. First create a bar code window in the Smart Forms form with 3 nodes as described in chapter [Datamatrix printing with Smart Forms](#)

. Only the first node have to be adapted to a new job. The node 2 and 3 remain unchanged.

(goto next page)

2. Enter the following program code:

Data declaration

```
data: inputdata type table of input_data with header line.  
clear inputdata.  
refresh inputdata.
```

Coding of 'MAC06' as function code

```
inputdata-dmvar = 'MAC06'.  
inputddata-dflag = 'F'.  
append inputdata.
```

Encoding of '12V + suplierno' as Text

```
concatenate '12V' suplierno into inputdata-dmvar.  
inputdata-dflag = 'T'.  
append inputdata.
```

Encoding of 'GS' as function code

```
inputdata-dmvar='GS'.  
inputddata-dflag = 'F'.  
append inputdata.
```

Encoding of '1P + partno suplierno' as Text

```
concatenate '1P' partno into dmvar.  
inputdata-dflag = 'T'.  
append inputdata.
```

Encoding of 'GS' als as function code

```
inputdata-dmvar5='GS'.  
inputddata-dflag = 'F'.  
append inputdata.
```

Encoding of 'S + serialo suplierno' as Text

```
concatenate 'S' serialno into dmvar.  
inputdata-dflag = 'T'.  
append inputdata.
```

Definition of bar code properties

```
read table inputdata index 1.  
inputdata-res = 150.  
inputdata-dmsize = 0.  
inputdata-x_dim = 4.  
modify inputdata index 1.
```

Passing the interface table inputdata to z_set_datamatrix

```
perform gen_datamatrix_sf in program z_set_datamatrix  
tables inputdata  
changing dmname.
```

.

In the implementation shown above, initially a type is assigned to each variable (eg **12 + SUPLIerno** defined as text). The separator "**GS**" was defined as a function code with the type designation **DFLAG = 'F'**. **MAC06** is also passed as function code with the type designation "**F**".

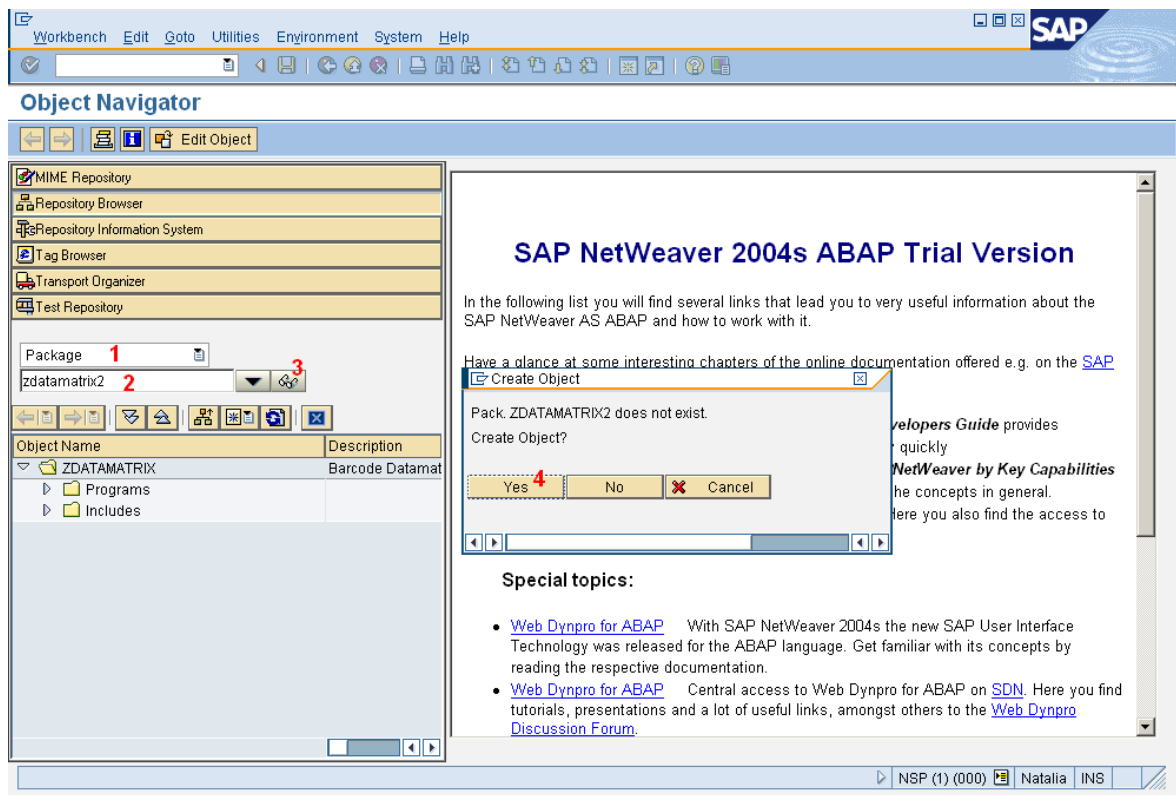
In the next step bar code properties with **RES**, **MSIZE** and **X_DIM** defined and written into the first record of interface table **INPUTDATA**.

The interface table **INPUTDATA** is passed to the form routine **gen_datamatrix_sf** in the program **z_set_datamatrix**.

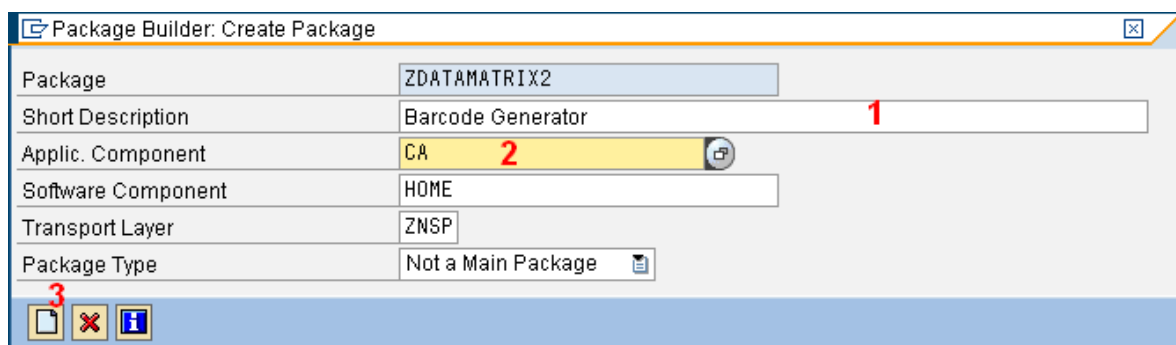
The return parameter is the name of the bar code **DMNAME**. The bar code is dynamically included into the form during further form processing.


8 Annex 1: Creating a new package (development class).

1. Start the transaction **SE80 (Object Navigator)**.
2. The following dialog appears:



3. Select the object „Package“ (1) (on older Systems "development class").
4. Enter the object name **ZDATAMATRIX** (2).
5. Click on the symbol "glasses" (3).
6. It appears a message with the information, that the package doesn't exist and asks you, if you want to create it. Click on **Yes** (4).
7. The following dialog appears:



8. Enter a short description (1).
9. Select „CA“ (2) as application component.
10. Click on the symbol  (3) to create the new package.

11. The prompt for transportable workbench request appears.

The dialog box has a title bar with a close button. It contains three input fields: 'Package' with the value 'ZDATAMATRIX2', 'Request' with the value 'NSPK900050' (highlighted in yellow), and 'Short Description' with the value 'RBarc2006'. Below the fields is a toolbar with a green checkmark, a floppy disk icon, a document icon, a button labeled 'Own Requests', and a red X icon. A red number '1' is placed above the document icon.

12. If no request for this installation has been created yet, create the request, otherwise go to pt. 16.

13. Click on the symbol  to create the new request.

The dialog box has a title bar with a close button. It contains several input fields: 'Request' with the value 'Workbench request', 'Short description' with the value 'Datamatrix' (highlighted in yellow), 'Project' (empty), 'Owner' with the value 'BCUSER', 'Status' with the value 'New', 'Last changed' with the value '04.05.2007 21:42:36', 'Source client' with the value '000', and 'Target' with the value 'CL5'. Below these fields is a 'Tasks' section with a list box containing 'User' and 'BCUSER'. At the bottom is a toolbar with a floppy disk icon, a document icon, and a red X icon. A red number '2' is placed above the floppy disk icon. A red number '1' is placed above the 'Short description' field.

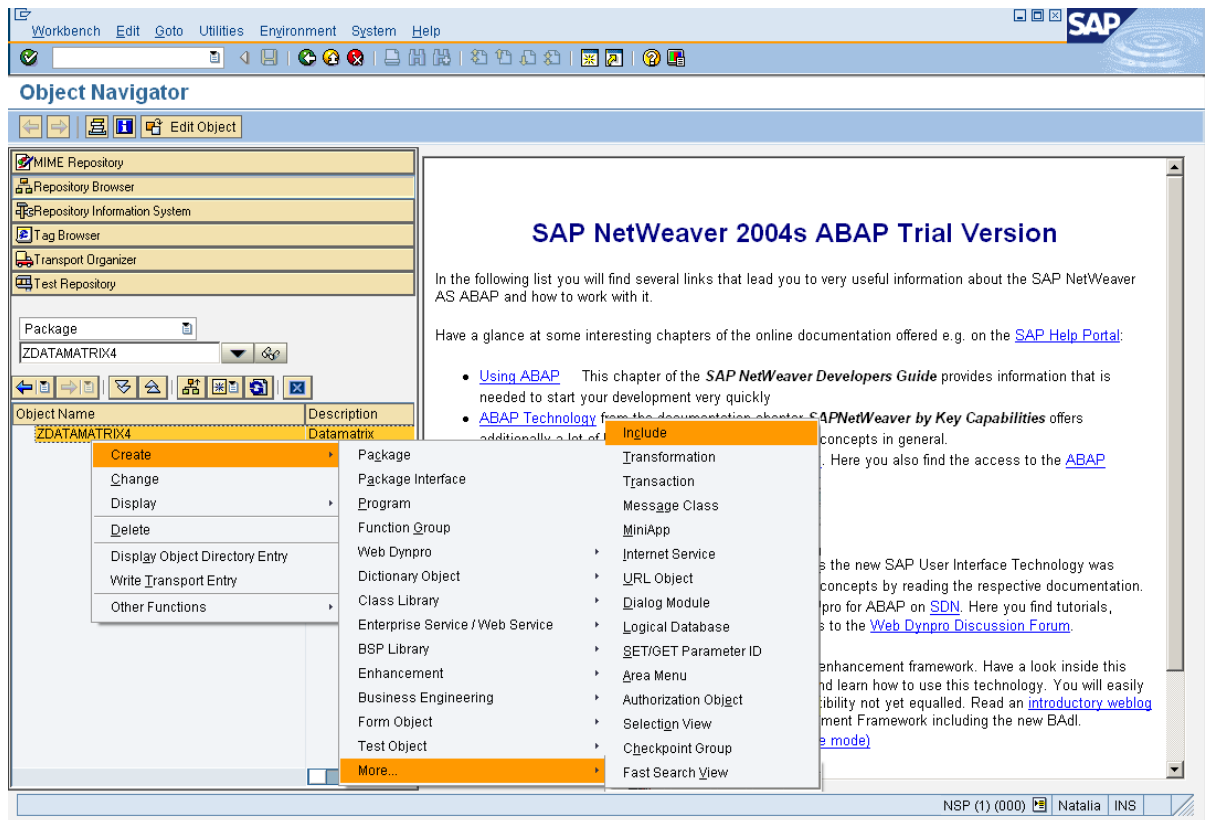
14. Enter a short description for the request (1).
15. Click on the floppy symbol (2) to save the request.
16. The request number appears in the field „Request“

The dialog box has a title bar with a close button. It contains three input fields: 'Package' with the value 'ZDATAMATRIX4', 'Request' with the value 'NSPK900053' (highlighted in yellow), and 'Short Description' with the value 'Datamatrix'. Below the fields is a toolbar with a green checkmark, a floppy disk icon, a document icon, a button labeled 'Own Requests', and a red X icon. A red number '1' is placed above the green checkmark.

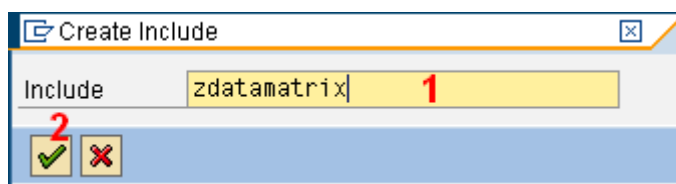
17. Click on the green check (1) to save the request.

9 Annex 2: Creating an include in the ABAP Workbench .

1. Start the **Object Navigator** (transaction **SE80**) and select the new Package (development class) **ZDATAMATRIX**.
2. Select the object name **ZDATAMATRIX** and click on the right mouse button.
3. Select from the context menu **Create/More/Include** (on older systems **Include** appears already on the second place).



4. Enter in the following dialog the name of the include, which has to be created (1) and click on the green check (2).



5. Click on the button "Save" (1) to create the include.

ABAP: Program Attributes ZDATAMATRIX2 Change

Title: Include ZDATAMATRIX2

Original language: EN English

Created: 04.05.2007 BCUSER

Last changed by:

Status: New(Revised)

Attributes

Type: INCLUDE program

Status:

Application:

Authorization Group:

☐ Editor lock

1 Save

6. Take care to save the include in the new package (1) and click on the floppy symbol (2) to save the include.

Create Object Directory Entry

Object: R3TR PROG ZDATAMATRIX2

Attributes

Package: ZDATAMATRIX4 1

Person Responsible: BCUSER

Original System: NSP

Original language: EN English

2 Local Object Lock Overview

7. The prompt for transportable workbench request appears. The request should be at this point already present, because it was created during saving the new Package (development class) **ZDATAMATRIX**. The request number should be the same as before (1).

Prompt for transportable Workbench request

Program: ZDATAMATRIX2

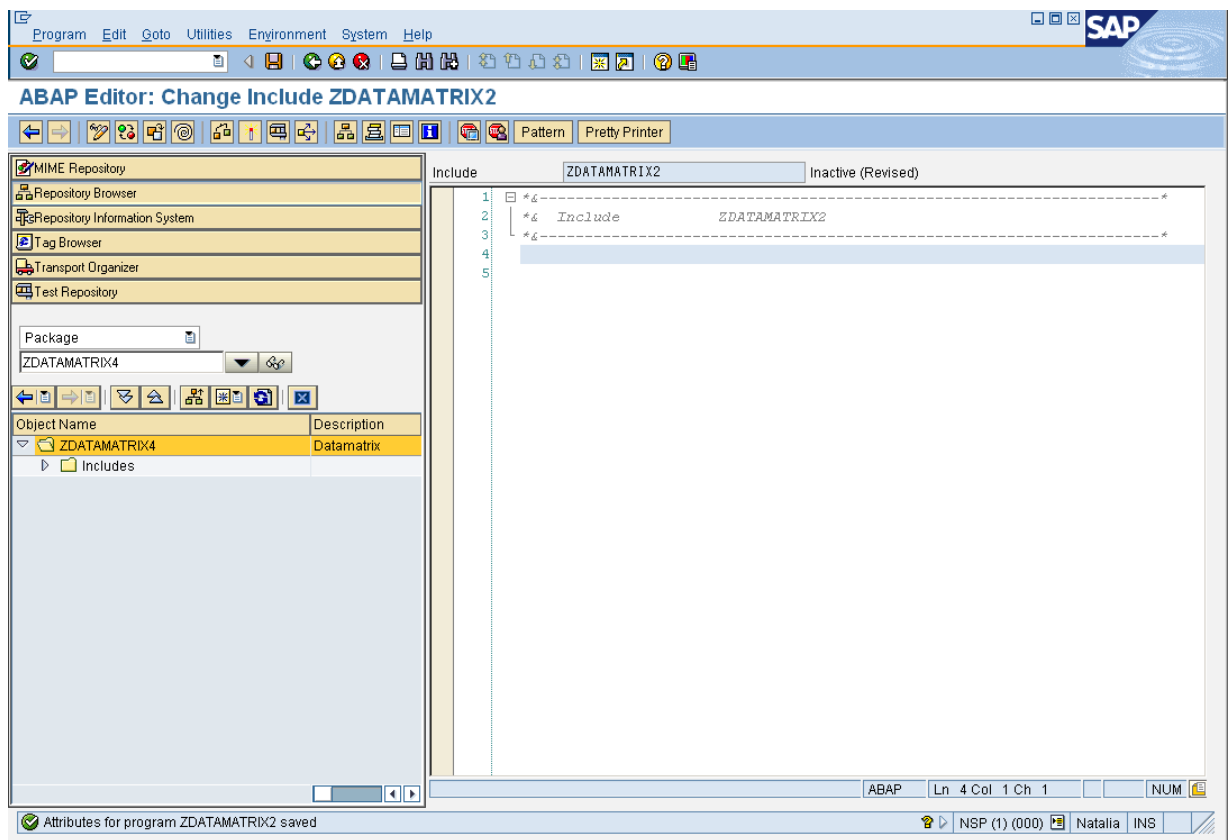
Request: NSPK900053 1 Workbench request

Short Description: Datamatrix

2

Own Requests

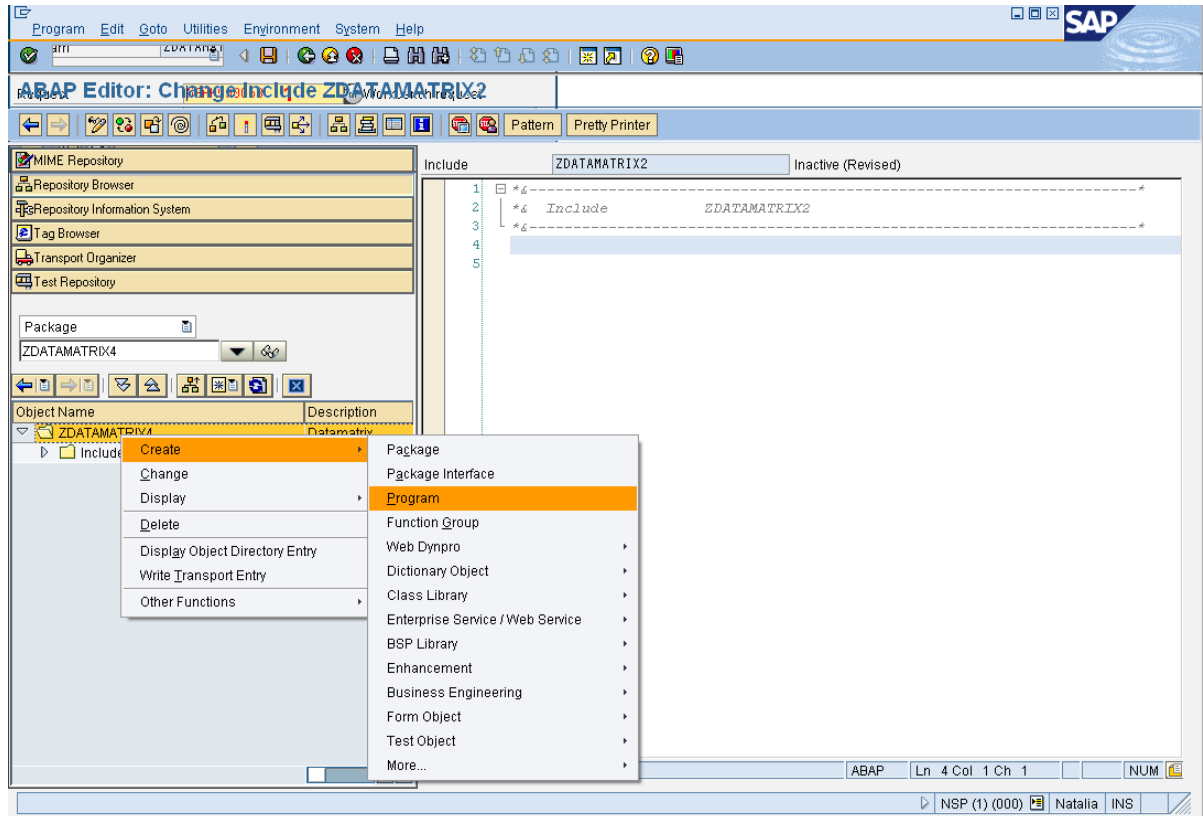
8. Click on the green check (2), to save the request.
9. Now the include will be finally saved and the window with the source code will appear.



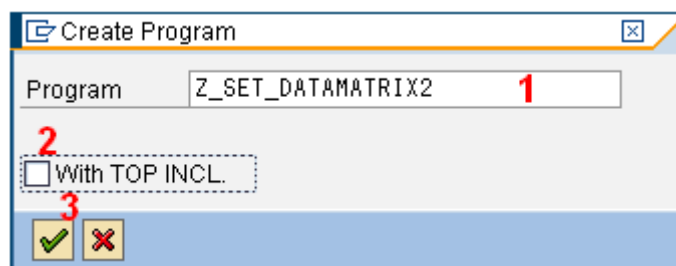
10. Delete the content of the source code, which was generated automatically.
11. Copy and paste the source code from the appropriate installation file and save the include. (go to page 54: [Annex 4: inserting content of installation files](#) into an ABAP Program.)

10 Annex 3: Creating a program in the ABAP workbench.

1. Start the **Object Navigator** (transaction **SE80**) and select the new Package (development class) **ZDATAMATRIX**.



2. Select the object **ZDATAMATRIX** and click on the right mouse button.
3. Select from the context menu **Create/Program**.
4. Enter in the next dialog the name of the new program (1), deselect the check box „With TOP-INCL.“ (2) and click on the green check (3).



5. Click on the button "Save" (1) to create the new program.

ABAP: Program Attributes Z_SET_DATAMATRIX2 Change

Title **Report Z_SET_DATAMATRIX2**

Original language **EN** English

Created **04.05.2007** **BCUSER**

Last changed by

Status **New(Revised)**

Attributes

Type **Executable program**

Status

Application

Authorization Group

Logical database

Selection screen

☐ Editor lock ☒ Fixed point arithmetic

☒ Unicode checks active ☐ Start using variant

1

Save

6. Take care to save the program in the new package (development class) **ZDATAMATRIX** (1) and click on the floppy symbol (2) to save the program.

Create Object Directory Entry

Object **R3TR PROG Z_SET_DATAMATRIX2**

Attributes

Package **ZDATAMATRIX4** **1**

Person Responsible **BCUSER**

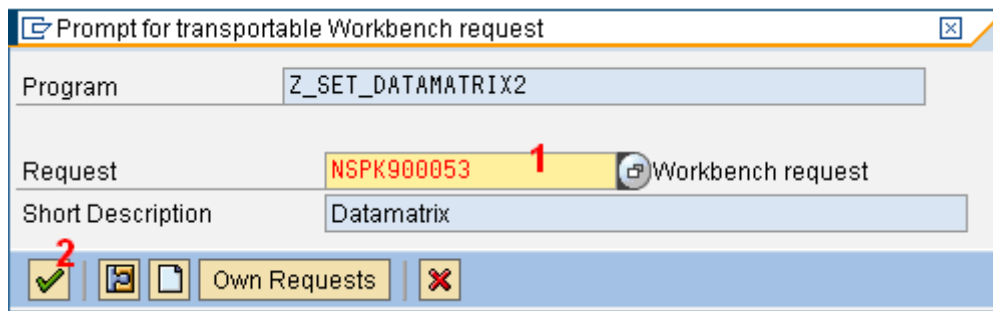
Original System **NSP**

Original language **EN** English

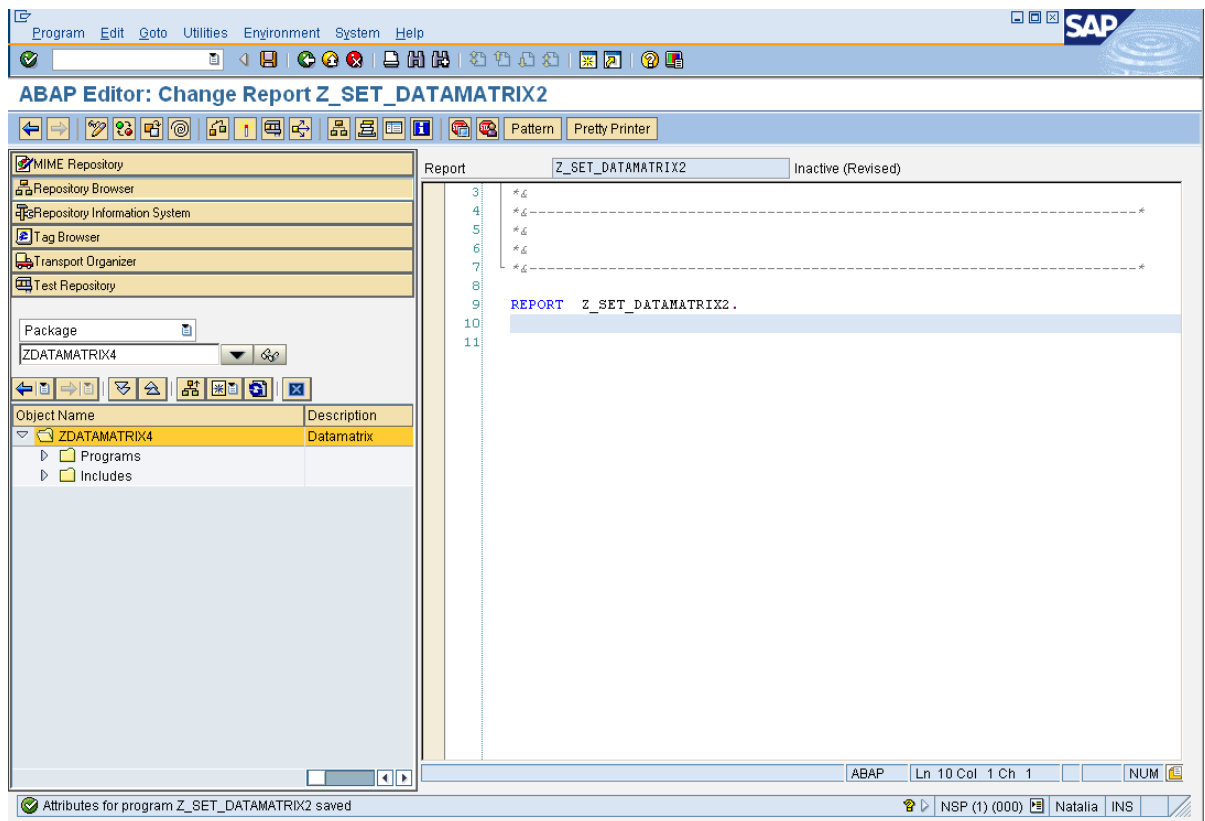
2

Local Object Lock Overview

7. The prompt for transportable workbench request appears. The request should be at this point already present, because it was created during saving the new Package (development class) **ZDATAMATRIX**. The request number should be the same as before (1)..



8. Click on the green check (2), to save the request.
9. Now the include will be finally saved and the window with the source code will appear.

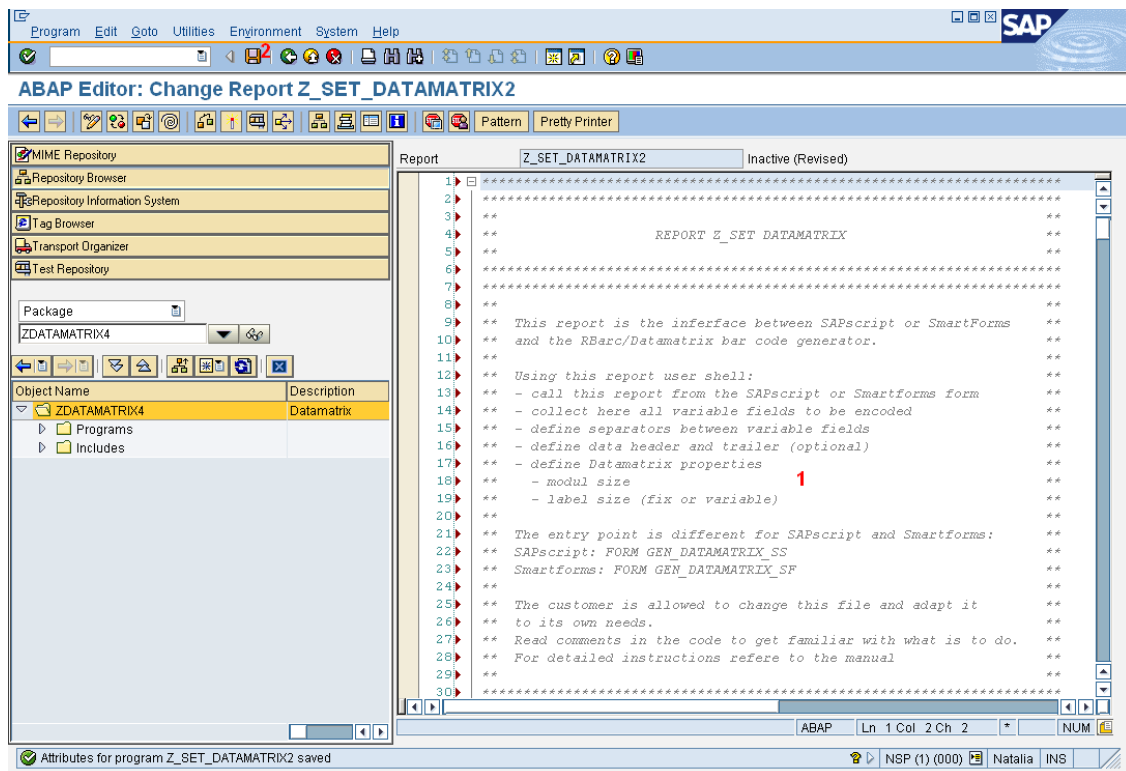


12. Delete the content of the source code, which was generated automatically.
10. Copy and paste the source code from the appropriate installation file and save the include.
(go to page 54: [Annex 4: inserting content of installation files](#) into an ABAP Program.)

11 Annex 4: inserting content of installation files into an ABAP Program.

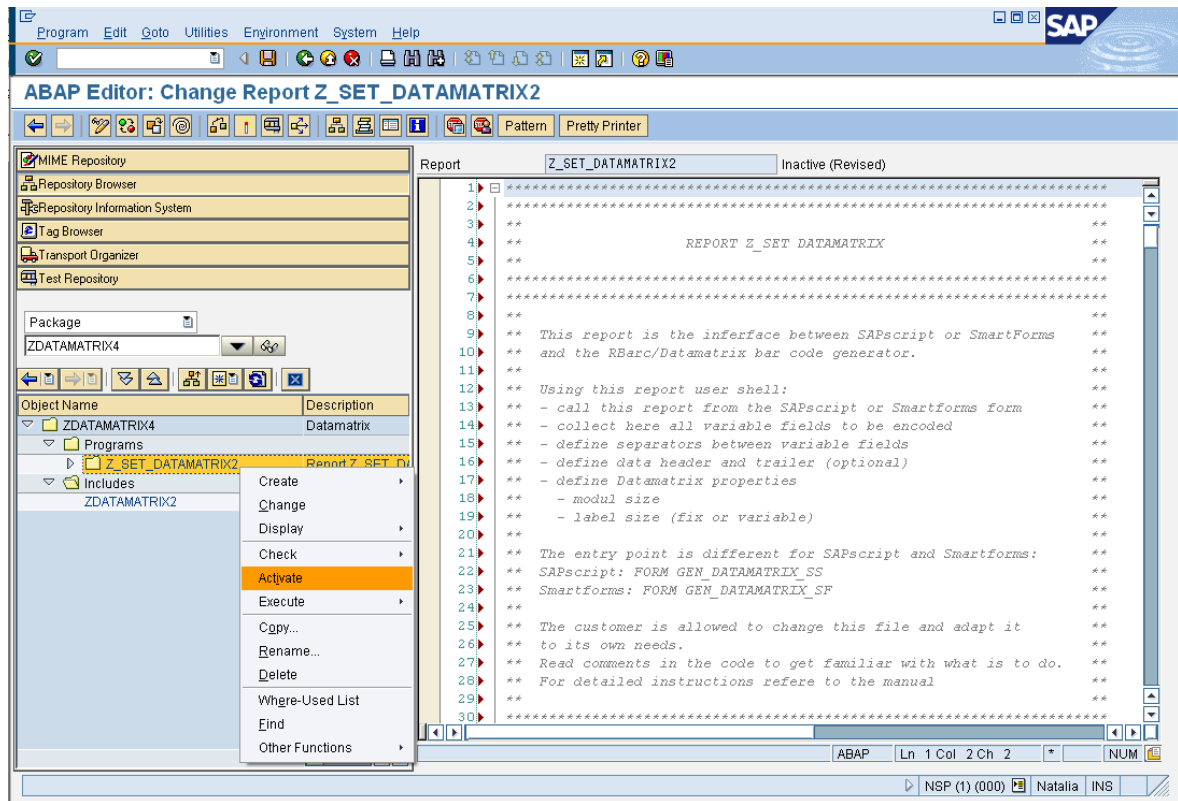
This procedure concerns as well programs as includes. As example we use **Z_SET_DATAMATRIX**.

1. Open the file **Z_SET_DATAMATRIX.PRG** from the subdirectory **\Install** with Windows Notepad (or other editor if another system as windows is used).
2. Select the content of the file with **Ctrl+A**.
3. Copy the selection into the clipboard by pressing **Ctrl+C**.
4. Change to SAPGui to the opened program **Z_SET_DATAMATRIX**.
5. Click with the left mouse button into the window with the ABAP source code (1) and paste the content of the clipboard by pressing **Ctrl+V**.
6. Click on the floppy symbol (2), to save the source code.

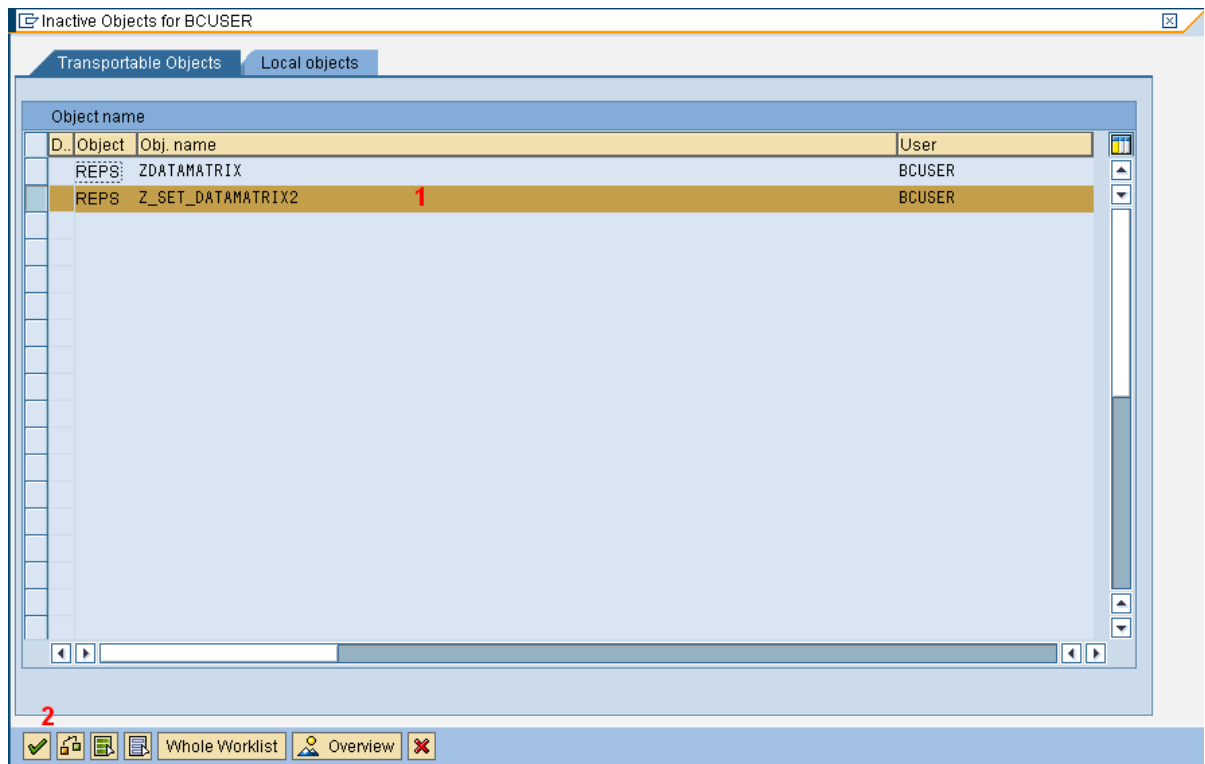


12 Annex 5: Activating of programs and includes

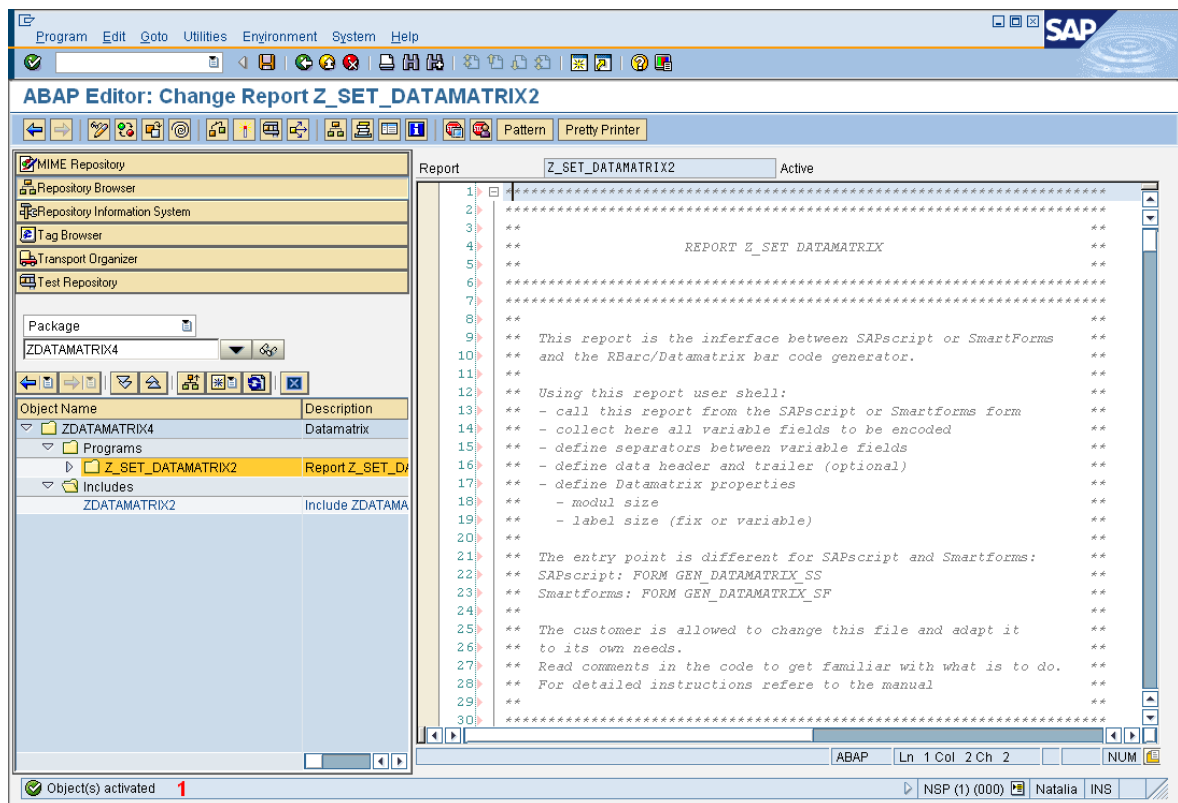
1. Open the **Object Navigator** (transaction **SE80**).
2. Select the object type „**Package**“
3. Enter the package name **ZDATAMATRIX** .
4. The tree nodes „**Programs**“ and „**Includes**“ appears.
5. Expand the tree node **Programs (1)** und **Includes (2)**.
6. Select the object, which shall be activated, click on the right mouse button and select „**Activate**“



7. The list of all inactive object appears. The object to be activated is already selected (1).click on the green check (2) to proceed the activating.

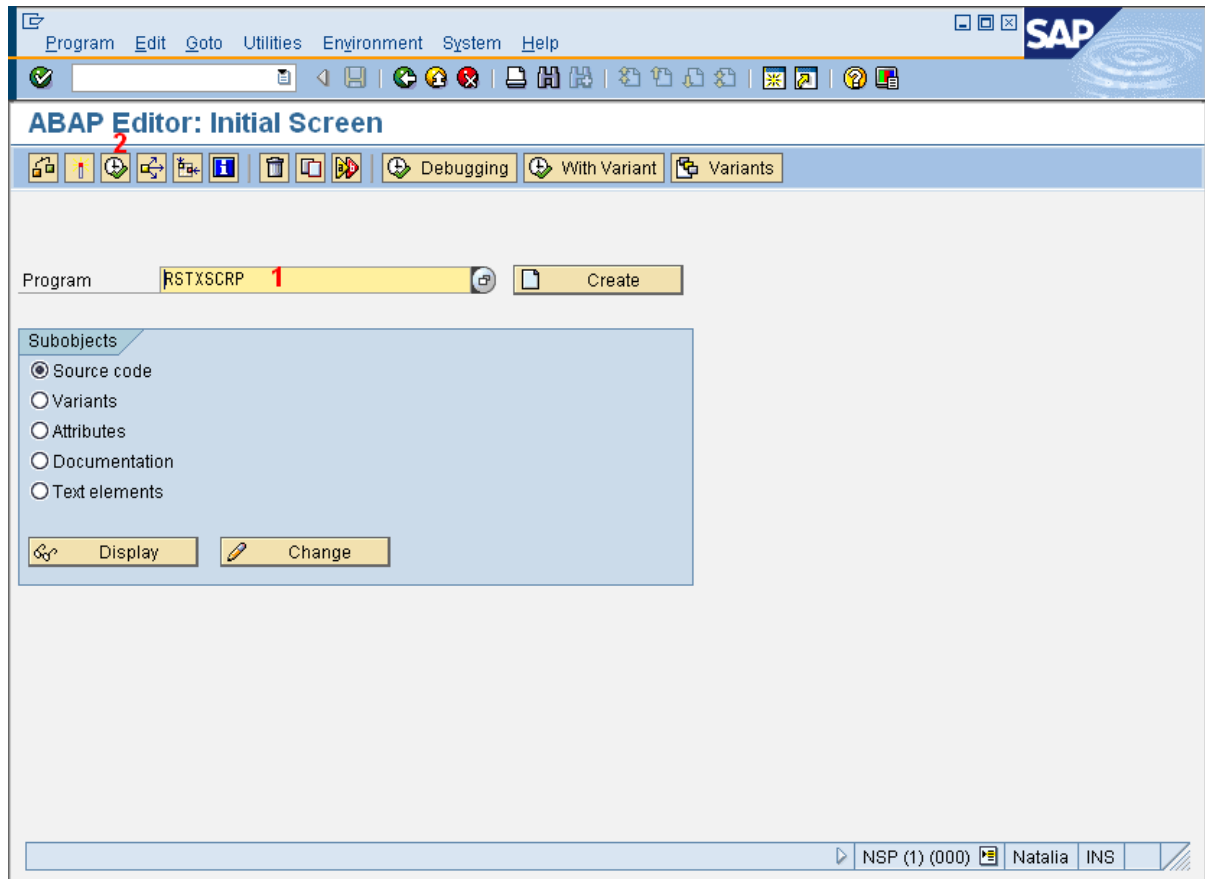


8. After the activating, a message appears in the status line (1). Programs can work only after a successful activating.



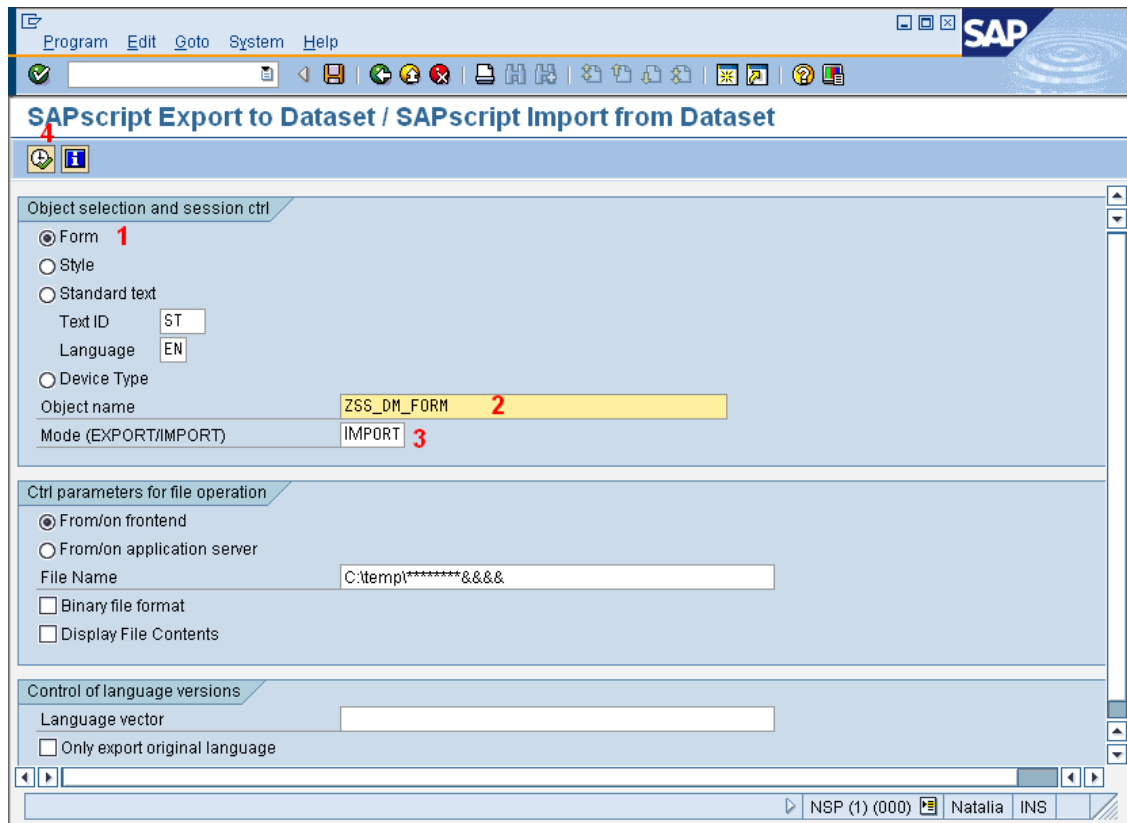
13 Annex 6: Executing of an ABAP program.

1. Start the transaction **SE38**.
2. Enter the name of the program in the field „**Program**“ (1).
3. Click on the execute symbol in the task bar (2).

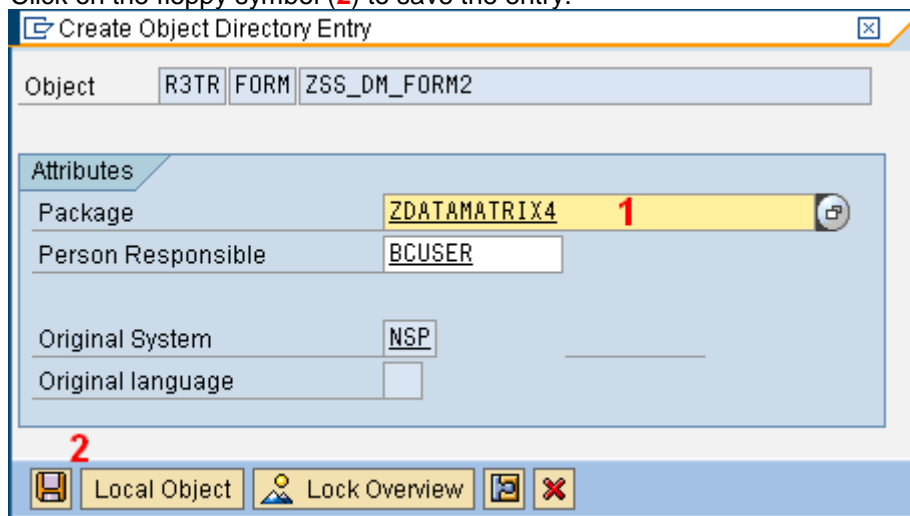


14 Annex 7: Importing of a SAPscript form .

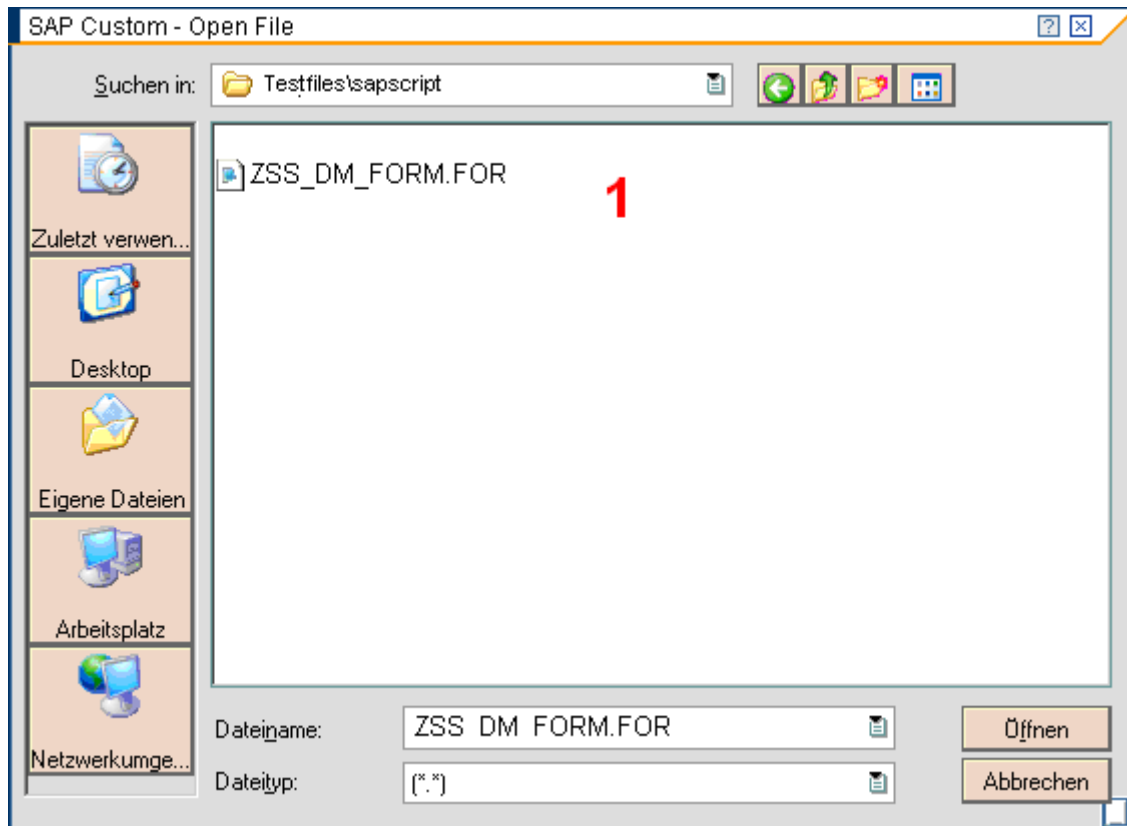
1. Start the SAP program **RSTXSCR**.
(Go to: [Annex 6: Executing of](#) an ABAP program)
2. Select the object type „**Form**“ (1).
3. Enter **ZSS_DM_FORM** in the field "Object name" (2).
4. Enter the mode „**IMPORT**“ (3).
5. Click on the symbol Execute (4)



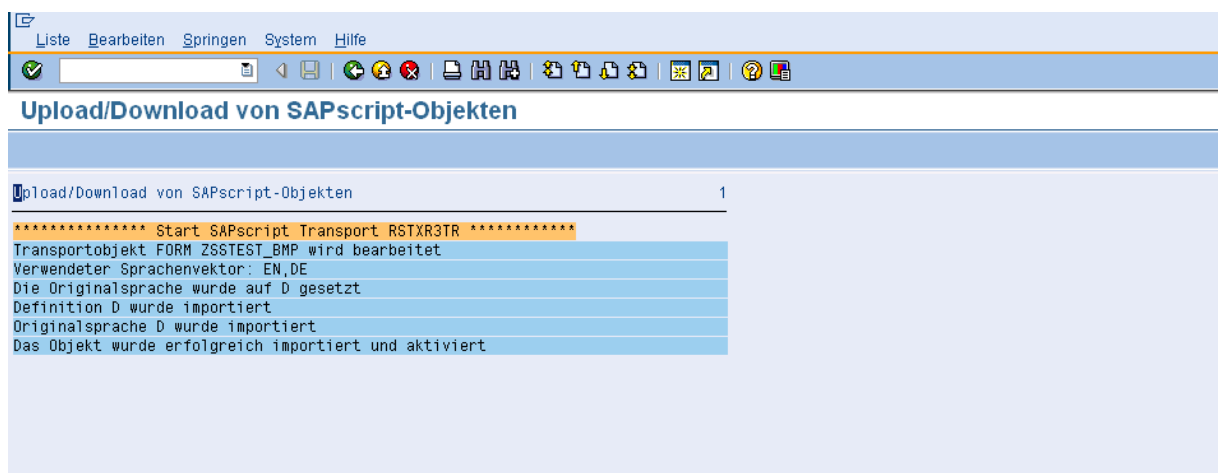
6. The dialog "Create Object Directory Entry" appears . Add the form to the package **ZDATAMATRIX** (1). Click on the floppy symbol (2) to save the entry.



7. The prompt for transportable workbench request papers. Use the existing request number (1) and click on the green check (2).
8. The file selection dialog appears. Select the file **ZSS_DM_FORM.FOR** from the subdirectory **\TestForms** (1) and click on the button **Open**.

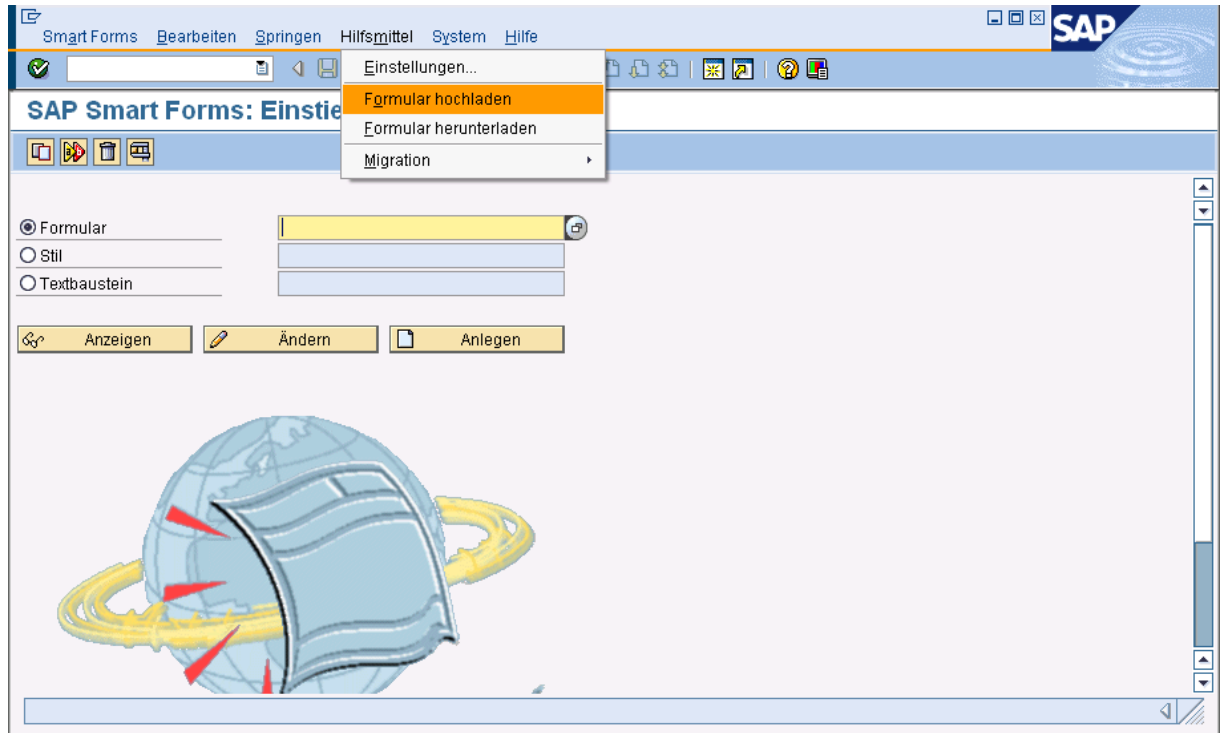


9. The form will be imported and at the end a message, similar to the following, appears:

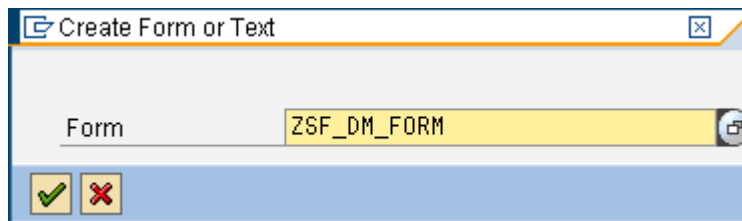


15 Annex 8: Uploading of a Smart Forms Form.

1. Start the transaction **Smartforms**.
2. Select **Utilities / Upload Form** from the menu.

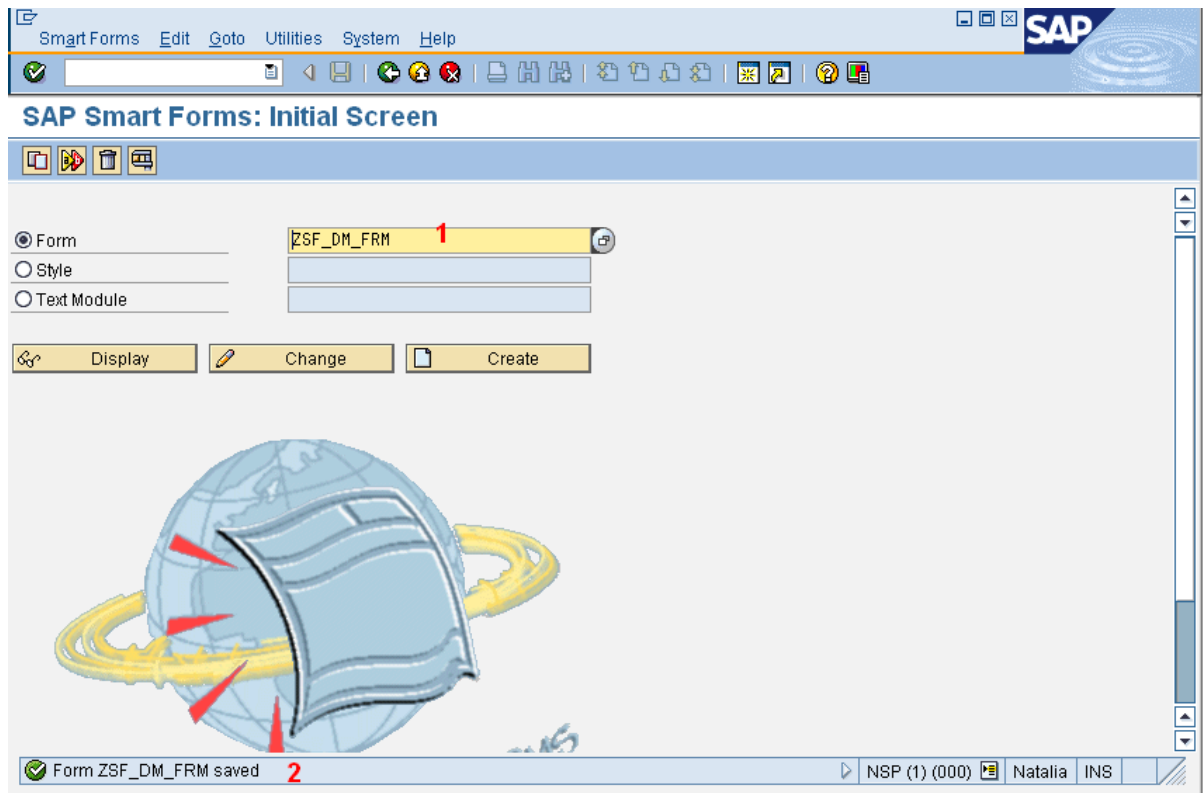


3. Enter the name of the form **ZSF_DM_FORM** in the next dialog and click on the green check.



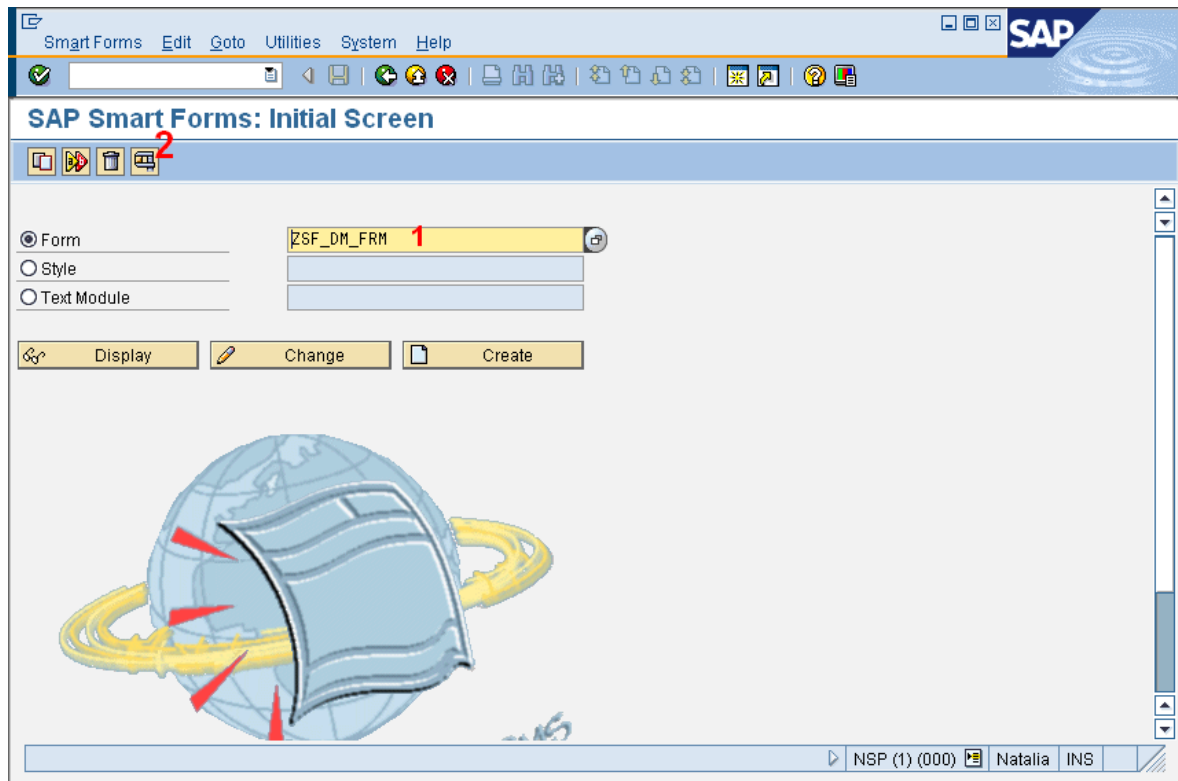
4. The file selection dialog appears. Select the file **ZSF_DM_FORM.XML** from the subdirectory **\TestForms** and click on the button **Open**.

5. The form will be uploaded. The name of the form appears automatically in the field "Form" (1) and a message in the status bar (2).

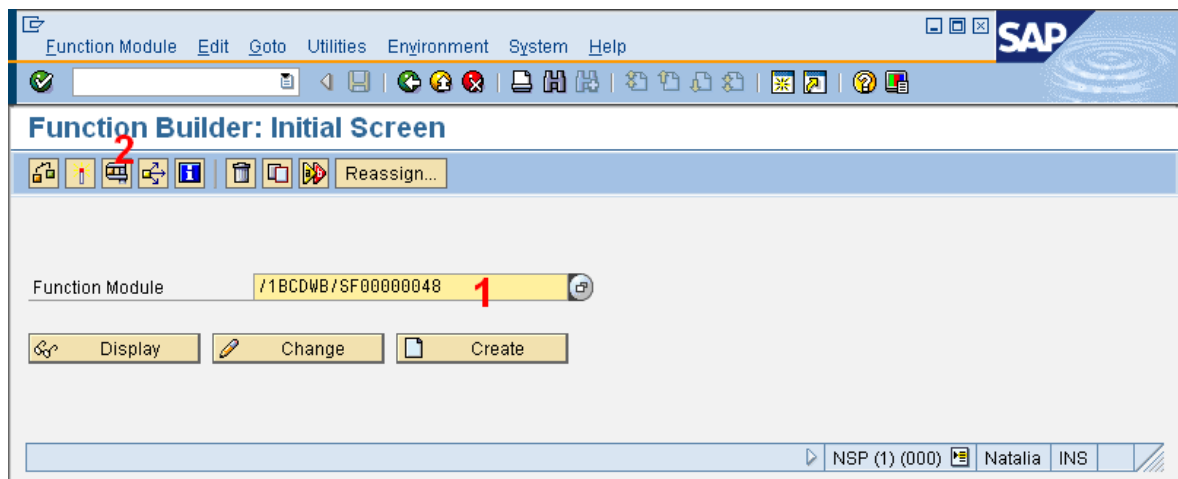


16 Annex 9: Testing a Smart Forms form.

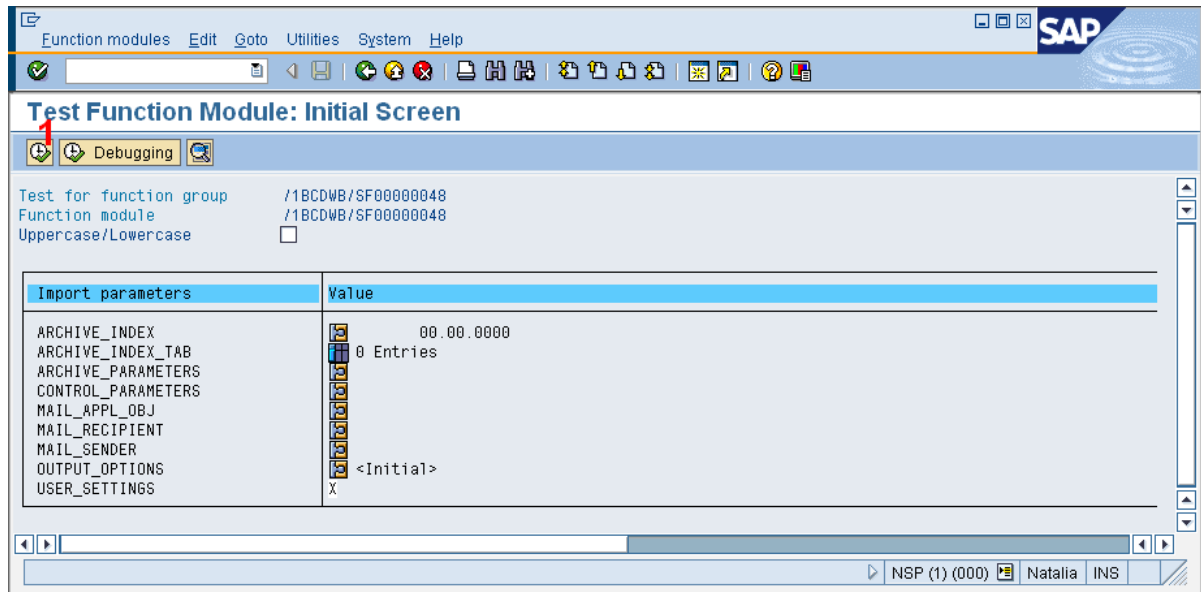
1. Start the transaction **Smartforms**.
2. Enter the name of the form **ZSF_DM_FORM** (1) and click on the test button (2).



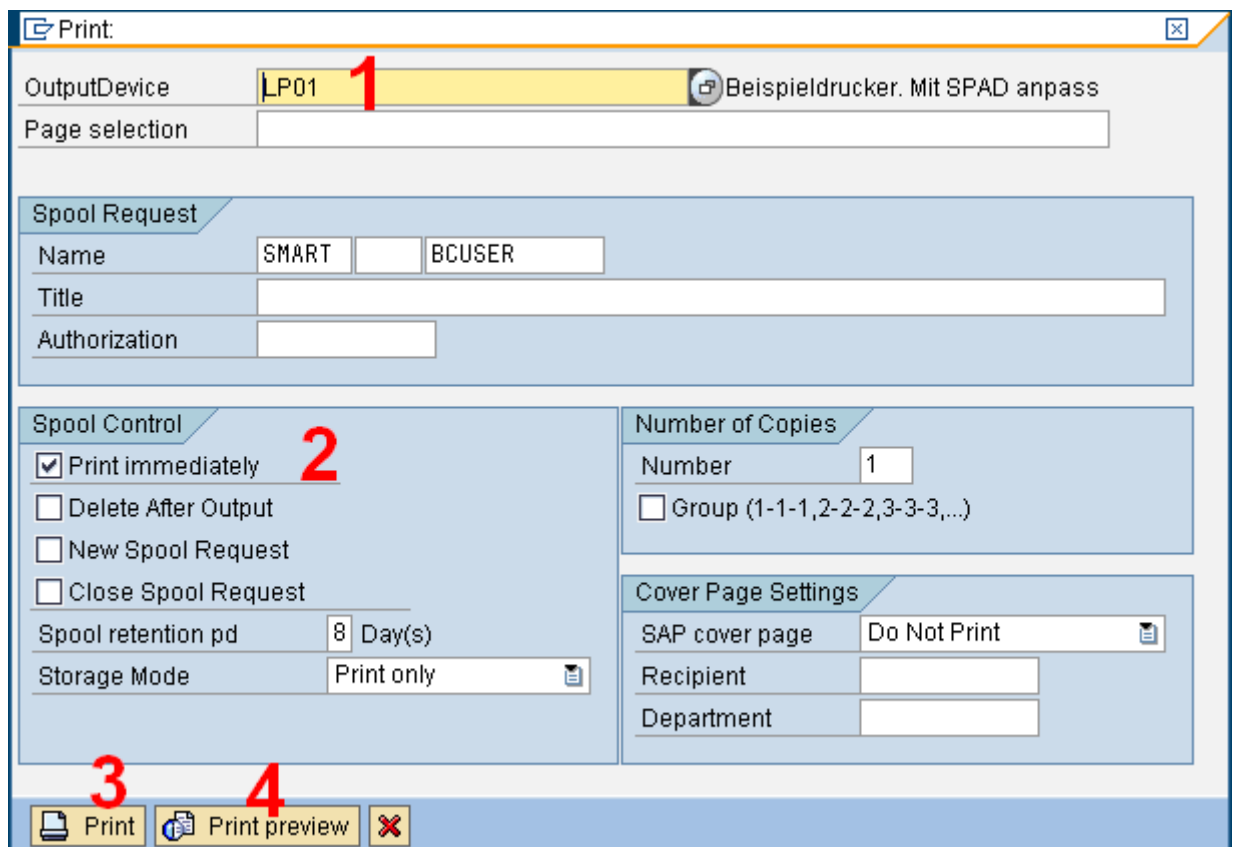
3. The system generates automatically a function, which name appears in the field "Function Module" (1) of the next dialog. Click against on the symbol **Test** (2).



4. Click on the button execute (1).

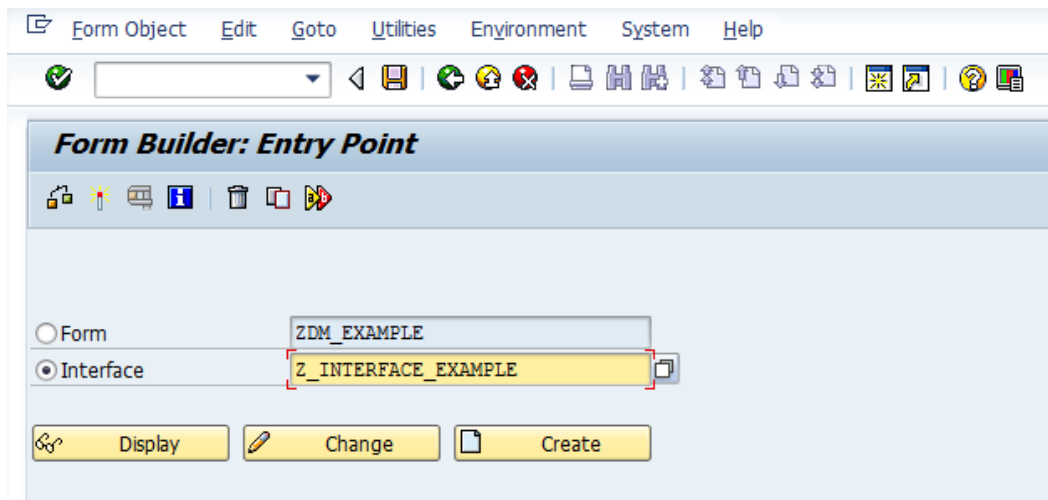


5. The print program starts now. Enter the name of the printer, where the form shall be printed out (1).
6. Check the box **Print immediately** (2)
7. Click on the button Print (3), to print the form on the printer, or on Print preview (4), to watch the form on the screen.

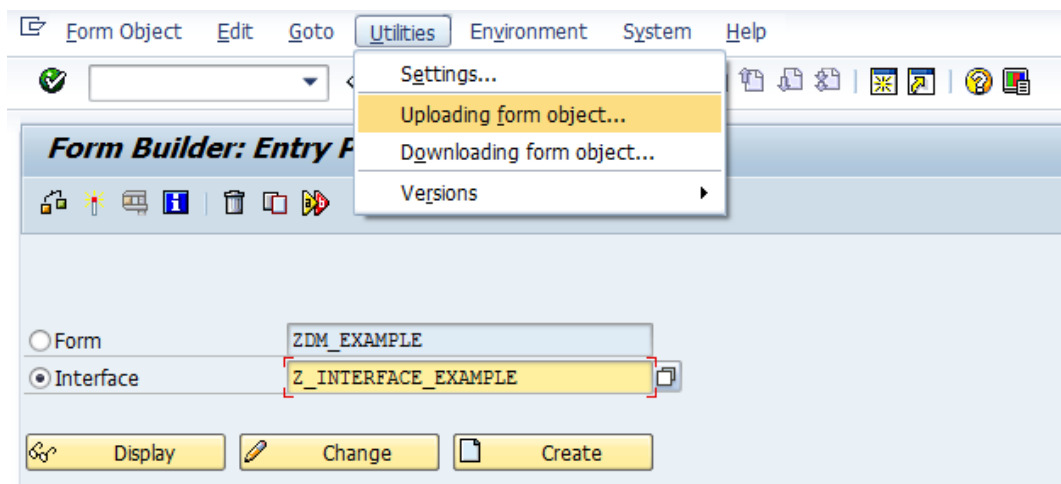


17 Annex 10: Upload an Interactive Forms Interface.

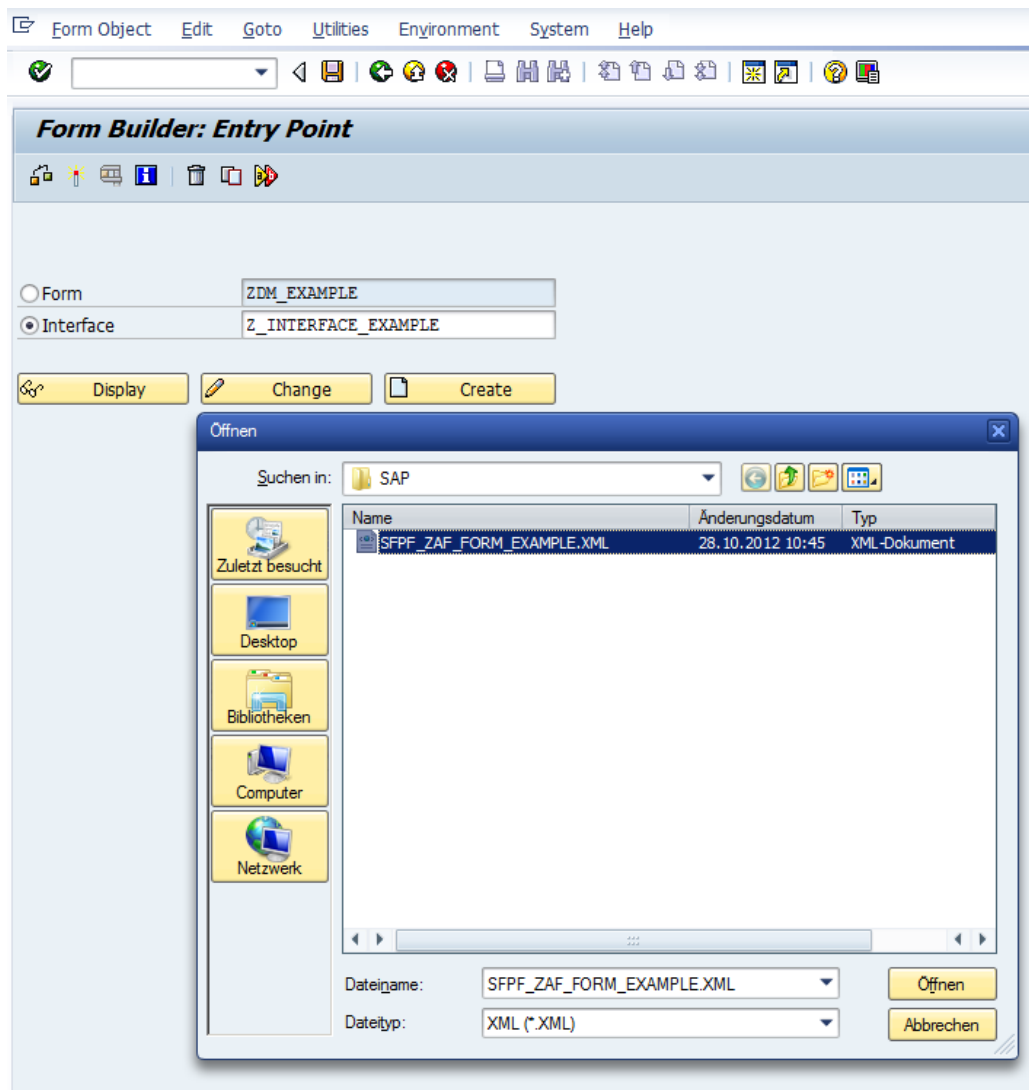
1. Start the FormBuilder by calling the transaction **SFP**.
2. Mark the radio button **Interface** and give the interface a name.



3. Select from the menu **Tool** → **Upload Form Object**.

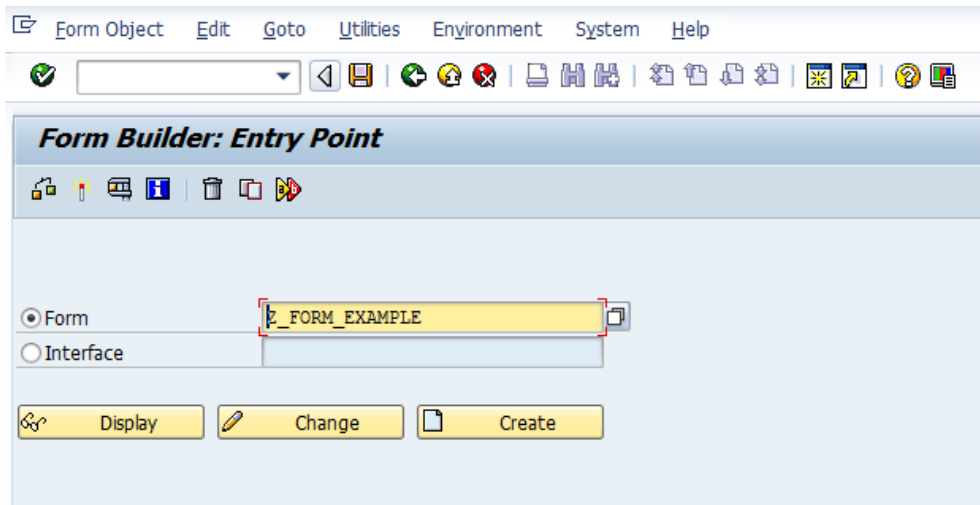


4. From the file selector, select the XML file and press the button **Open**.

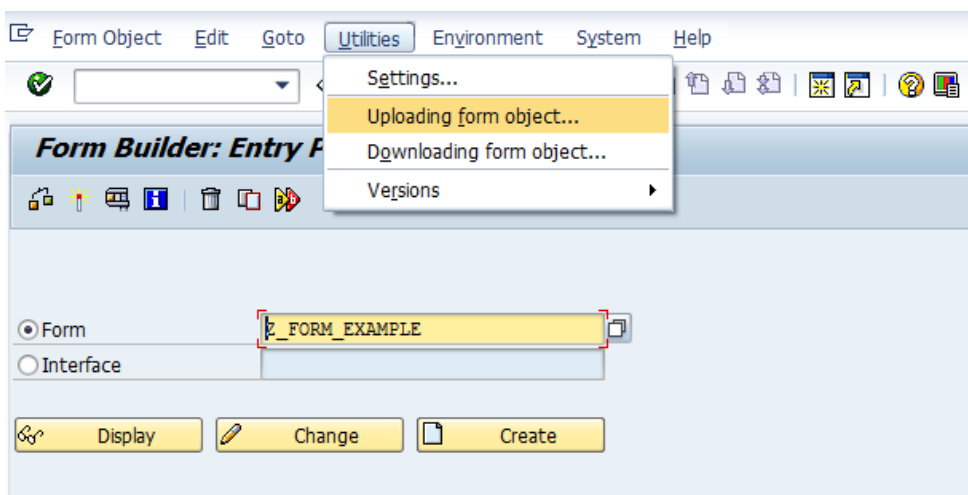


18 Annex 11: Upload an Interactive Forms Form.

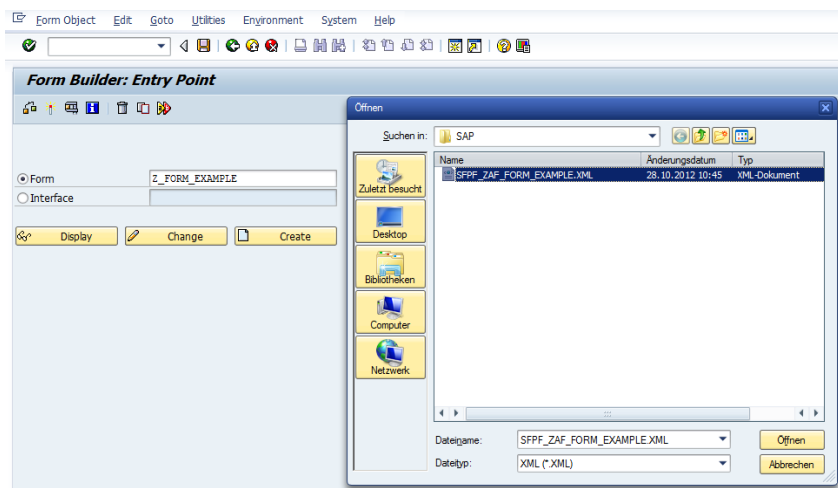
1. Start the FormBuilder by calling the transaction **SFP**.
2. Mark the radio button **Form** and give the form a name.



3. Select from the menu **Tool** → **Upload Form Object**.

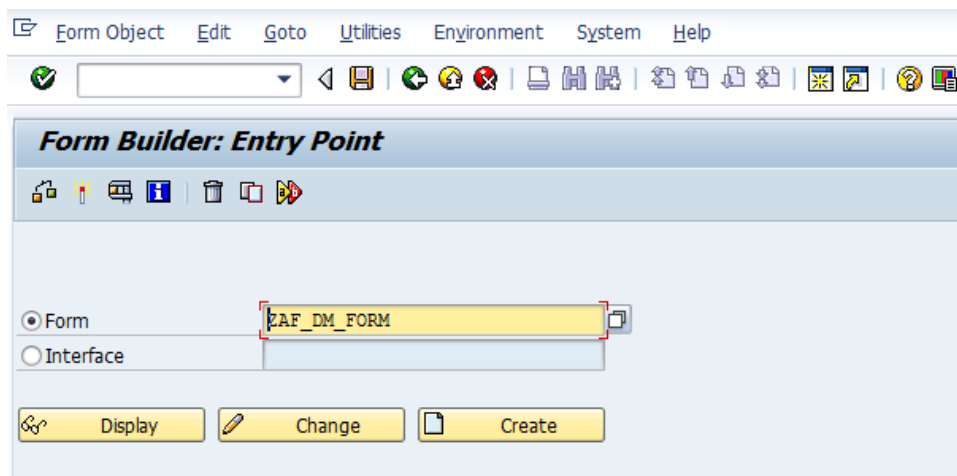


4. From the file selector, select the XML file and press the button **Open**.

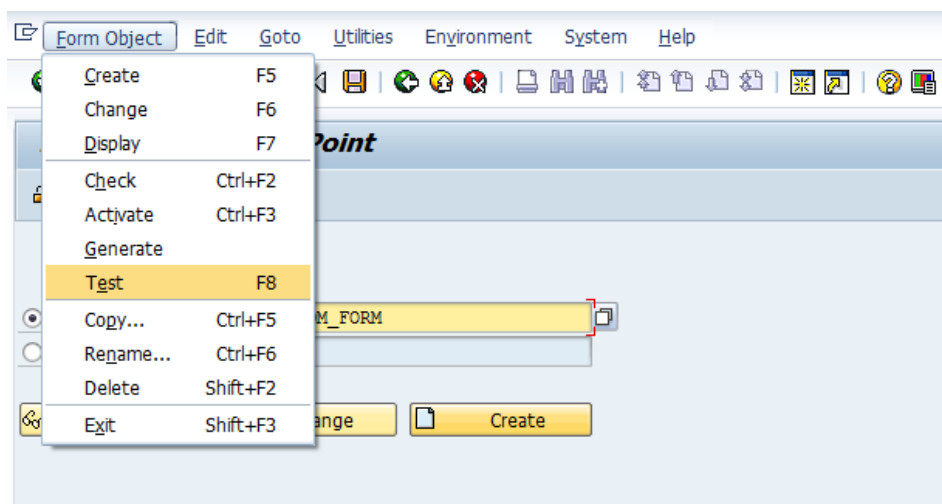


19 Annex 12: Test an Interactive Forms Form

1. Start the FormBuilder by calling the transaction **SFP**.
2. Mark the radio button **Form** and enter the name of the form (in the example: **ZAF_DM_FORM**).

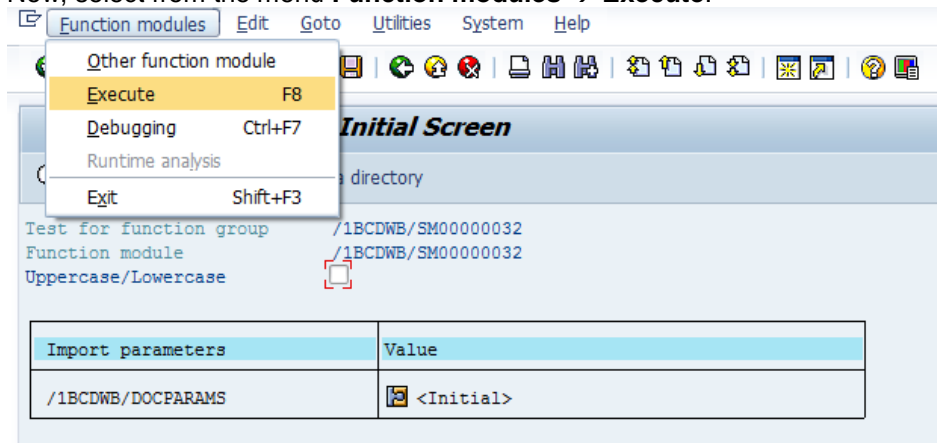


3. Select from the menu **Form Object** → **Test**.



The generated function block is now shown.

4. Now, select from the menu **Function modules** → **Execute**.



5. Enter a printer that was not allocated with an SAPWIN or SWIN device type and press the button **Print View Display**.

System Help

SAP

Print:

Output Device: LOCA

Spool Request

Name: PBFORM LOCA XXXX

Cover Page Text:

Authorization:

Spool Control

☒ Print Immediately

☒ Delete After Output

☐ New Spool Request

☐ Close Spool Request

Spool Retention Per.: 8 Day(s)

Storage Mode: 1 Print only

Number of Copies

Number of Copies: 1

Cover Page Settings

SAP Cover Page: Do Not Print

Recipient(s):

Department:

Print Print Preview

6. It should now be possible to view the Print View Display.



Text Edit Goto Extras System Help

Print Preview of LP01 Page 00001 of 00001

Archive Print and Archive

RBarc/Datamatrix

2D Barcode GS1 Datamatrix ECC 200 generated on a SAP System
A solution from Suchy MIPS www.suchymips.de

This demo presents 2 Datamatrix-Labels with identical content but encoded in different ways.

Both Datamatrix codes encode following data: '[(>[RS]06[GS]2L'

[RS] means the function code for Record Separator (ASCII Value 30 dec)
[GS] means the function code for Group Separator (ASCII Value 29 dec)

The left Datamatrix-Code was encoded using different variables with different Data Type Flags:

dmvar1 = '(>'. dflag = 'T'.
dmvar2 = 'RS'. dflag = 'F'.
dmvar3 = '06'. dflag = 'T'.
dmvar4 = 'GS'. dflag = 'F'.
dmvar5 = '2L'. dflag = 'T'.

The right Datamatrix-Code was encoded using one variable which encodes all data in ASCII-HEX.

dmvar1 = '5B283E1E30361D324C'.
DFLAG = 'X'.

Available Data Type Flags for data encoding are:

'T' - any text
'X' - data encoded as an ASCII-Hex string (e.g. 5B283E1E...)
'F' - function codes like 'GS' 'RS' 'EOT'
'B' - Byte with values 0 - 255
'L' - ECI (Extended Channel Interpretation)

For more details refer to the manual.

SAP