



How to Make Your ABAP Code Unicode- Enabled

Dr. Christian Hansen
Server Technology Internationalization,
SAP AG

THE BEST-RUN BUSINESSES RUN SAP





Part I – SAPs approach to Unicode

- Demo – Unicode vs. Non-Unicode R3
- Unicode Essentials
- Transparent Unicode Enabling for R/3

Part II – Unicode Enabled ABAP

- Unicode Restrictions
- New ABAP Features

Part III – Tools for Unicode Enabling

- Migration to Unicode
- Unicode Scan UCCHECK
- Coverage Analyzer SCOV

Exercises



What is Unicode?

- Character encoding schema for (nearly) all characters used world wide

[ما هي الشفرة الموحدة "يونيكود"؟](#) in Arabic

[什麼是Unicode\(統一碼/標準萬國碼\)?](#) in Chinese (Traditional)

[What is Unicode?](#) in English

[რა არის უნიკოდი?](#) in Georgian

[Τι είναι το Unicode?](#) in Greek

[यूनिकोड क्या है?](#) in Hindi

[Cos'è Unicode?](#) in Italian

[ユニコードとは何か？](#) in Japanese

[유니 코드에 대해?](#) in Korean

[Что такое Unicode?](#) in Russian

- Each character has a unique number („Unicode code point“)
 - ◆ Notation U+nnnn (where nnnn are hexadecimal digits)
- See <http://www.unicode.org> for complete code charts

Demo – View table content in Old MDMP World

West European View

COLOR	SPRAS	NAME
B	DE	blau
G		grün
R		rot
Y		gelb
B	EN	blue
G		green
R		red
Y		yellow
B	JA	青
G		緑
R		赤
Y		黄
B	KO	파란색
G		초록색
R		빨간색
Y		노란색

Japanese View

COLOR	SPRAS	NAME
B	DE	blau
G		grün
R		rot
Y		gelb
B	EN	blue
G		green
R		red
Y		yellow
B	JA	青
G		緑
R		赤
Y		黄
B	KO	파란색
G		초록색
R		빨간색
Y		노란색

Korean View

COLOR	SPRAS	NAME
B	DE	blau
G		grün
R		rot
Y		gelb
B	EN	blue
G		green
R		red
Y		yellow
B	JA	파란색
G		초록색
R		빨간색
Y		노란색

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



- In an MDMP (Multi Display Multi Processing Code Page) system character data is encoded in different code pages. Depending on the view you choose, you will handle only a part of the data correctly. In an MDMP system you have to take care of the proper use of language keys.
- Byte representation of the character data in the database and in the memory of the application server:

- German	: rot	x'726F74'	ISO-8859-1
- English	: red	x'726564'	ISO-8859-1
- Korean	: 빨간색	x'BBA1B0A3BBF6'	KSC5601
- Japanese	: 赤	x'90D4'	SJIS

Demo - View table content on Unicode System

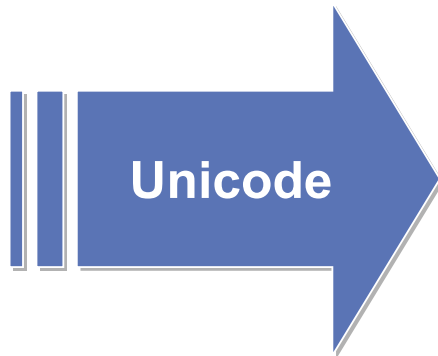


Table entry Edit SAP

Data Browser: Ta

COLOR	SPRAS	NAME
B	DE	blau
G	DE	grün
R	DE	rot
Y	DE	gelb
B	EN	blue
G	EN	green
R	EN	red
Y	EN	yellow
B	JA	青
G	JA	緑
R	JA	赤
Y	JA	黄
B	KO	파란색
G	KO	초록색
R	KO	빨간색
Y	KO	노란색

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



- In a Unicode system the character data is encoded in only one code page. There is only one view: Missinterpretation of the data is not possible.
- Byte representation of the character data in the memory of the application server with the UTF-16 variant of Unicode:
 - German : rot x'0072006F0074' UTF-16 Big Endian
 - English : red x'007200650064' UTF-16 Big Endian
 - Korean : 빨간색 x'BE68AC04C0C9' UTF-16 Big Endian
 - Japanese : 赤 x'8D64' UTF-16 Big Endian
- The database may contain the same byte representation or any 1:1 transformation into any other Unicode representation (UTF-8, UTF-16 Big Endian, UTF-16 Little Endian)

■ Representation of Unicode Characters

UTF-16 – Unicode Transformation Format, 16 bit encoding

- Fixed length, 1 character = 2 bytes (surrogate pairs = 2 + 2 bytes)
- Platform dependent byte order
- 2 byte alignment restriction

UTF-8 – Unicode Transformation Format, 8 bit encoding

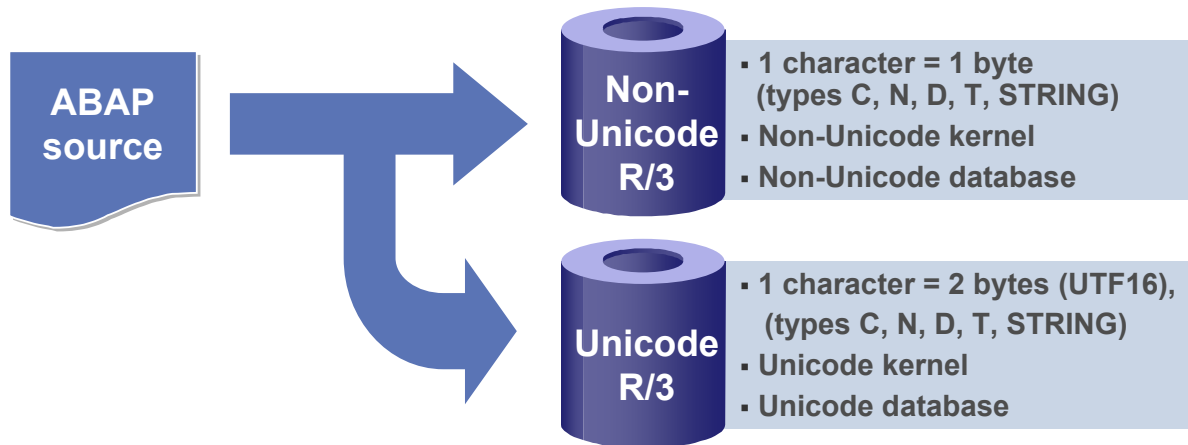
- Variable length, 1 character = 1...4 bytes
- Platform independent
- no alignment restriction
- 7 bit US ASCII compatible

Character	Unicode codepoint	UTF-16 big endian	UTF-16 little endian	UTF-8
a	U+0061	00 61	61 00	61
ä	U+00E4	00 E4	E4 00	C3 A4
α	U+03B1	03 B1	B1 03	CE B1
𐀀	U+3479	34 79	79 34	E3 91 B9

■ Transparent Unicode Enabling of R/3

Character Expansion Model

- Separate Unicode and non-Unicode versions of R/3



- No explicit Unicode data type in ABAP
- Single ABAP source for Unicode and non-Unicode systems

Implications:

- Major part of ABAP coding is ready for Unicode without any changes
- Minor part of ABAP coding has to be adapted to comply with Unicode restrictions
 - ◆ Syntactical restrictions
 - ◆ Additional runtime checks
 - ◆ Runtime tests for semantic changes

■ Unicode-Enabled ABAP Programs

Program attribute „Unicode checks active“

- Required to run on a Unicode system

	Non-Unicode system	Unicode system
Attribute set (Unicode enabled)	ok	ok
Attribute not set (not Unicode enabled)	ok	not allowed

- If attribute is set, additional restrictions:
 - ◆ apply at compile and at run time
 - ◆ apply in Unicode systems and in non-Unicode systems
 - ◆ ensure that program will run on non-Unicode and Unicode systems with (almost) identical behavior

- Attribute can be set for all program objects, i.e. reports, function pools, classes, interfaces, type pools, ...

Program Attribute „Unicode checks active“

Title	Example: Unicode enabled program		
Original language	EN	English	
Created	13.08.2001	SCHIED	
Last changed by	13.08.2001	SCHIED	
Status	Inactive		

Attributes			
Type	Executable program		
Status			
Application			
Authorization Group			
Development class	\$TMP	Private Test Programs and Utilities	
Logical database			
Selection screen			
<input type="checkbox"/> Editor lock	<input checked="" type="checkbox"/> Fixed point arithmetic		
<input checked="" type="checkbox"/> Unicode checks active	<input type="checkbox"/> Start using variant		

Save



Contents

Part I – SAPs approach to Unicode

- Demo – Unicode vs. Non-Unicode R3
- Unicode Essentials
- Transparent Unicode Enabling for R/3

Part II – Unicode Enabled ABAP

- Unicode Restrictions
- New ABAP Features

Part III – Tools for Unicode Enabling

- Migration to Unicode
- Unicode Scan UCCHECK
- Coverage Analyzer SCOV

Exercises

Design Goals

- Platform independence
 - ◆ Identical behavior on Unicode and non-Unicode systems
- Highest level of compatibility to the pre-Unicode world
 - ◆ Minimize costs for Unicode enabling of ABAP Programs
- Improved security, maintainability, and readability of ABAP programs

Main Features

- Clear distinction between character and byte processing
 - 1 Character <> 1 Byte**
- Enhanced checks prevent programming based on memory layout assumptions
- Improved conversion facilities
- Improved dataset interface
- Improved support for dynamic programming

Character Processing

```
CONCATENATE cf1 cf2 TO cf3.  
IF cf1 CS cf2. ...
```

- String operations are only allowed for character-like operands
 - ◆ ABAP types C, N, D, and T, STRING
 - ◆ Structures consisting only of characters (C, N, D, T)
 - ◆ X and XSTRING are no longer considered character-like types

Byte Processing

```
CONCATENATE xf1 xf2 TO xf3 IN BYTE MODE.  
IF xf1 BYTE-CS xf2. ...
```

- Variants of string operations for byte processing
 - ◆ Addition „IN BYTE MODE“ for statements
 - ◆ Prefix „BYTE-“ for comparison operations
- Only operands of type X or XSTRING allowed

Unicode Restrictions – Length and Distance

Determining the Length and Distance

- Counted in bytes or in characters? Specify!

DESCRIBE FIELD...LENGTH... IN (BYTE | CHARACTER) MODE.

DESCRIBE DISTANCE BETWEEN ... AND ... INTO ...
IN (BYTE | CHARACTER) MODE.

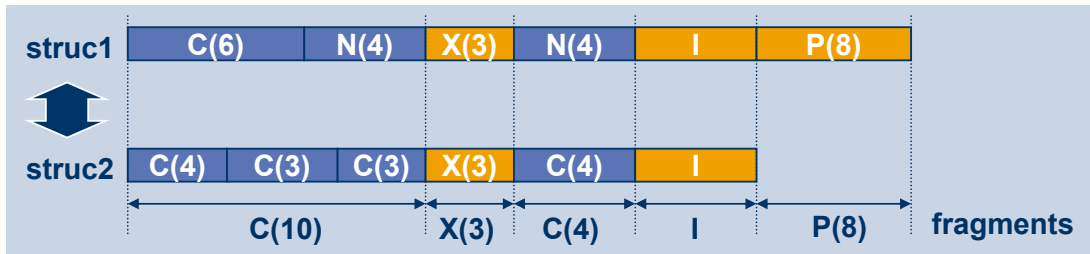
Example

```
FORM write3 USING fld TYPE c.  
  DATA: fldlen TYPE i.  
  DESCRIBE FIELD fld LENGTH fldlen IN CHARACTER MODE.  
  IF fldlen >= 3.  
    WRITE: / fld(3).  
  ENDIF.  
ENDFORM.
```

Unicode Restrictions - MOVE

MOVE Between Incompatible Structures

- Matching data layout („fragment views“) required



Example

```
DATA:
  BEGIN OF cstru,
    first(10) TYPE c,
    tab(1)    TYPE c,
    last(10)  TYPE c,
  END OF cstru.
```

cstru = xstru.

```
DATA:
  BEGIN OF xstru,
    first(10) TYPE c,
    tab(1)    TYPE x VALUE '09',
    last(10)  TYPE c,
  END OF xstru.
```

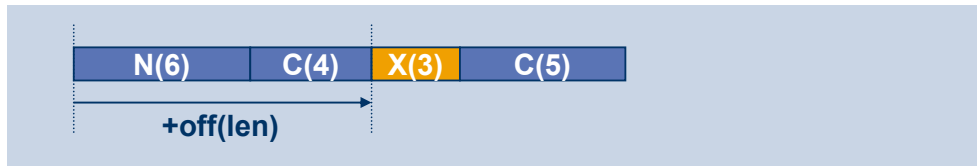
"Unicode error!"

- Same rule as for MOVE applies for implicit Moves too:
 - Operations for internal tables: LOOP AT itab INTO wa ...
 - Comparison between structures
 - Moving/Comparing Internal Tables: rules based on line types of tables
 - Database Operations with workareas:
 - SELECT * FROM dbtab INTO wa ...
 - SELECT * FROM dbtab INTO TABLE wa ...
 - UPDATA dbtab FROM wa ...
 - ...

Unicode Restrictions – Access with Offset or Length

Access To Structures With Offset/Length

- Structure must begin with characters
- Offset/length counted in characters
- Access only allowed within the character type prefix of a structure



ASSIGN fld+off(len) TO ...

- Access must not exceed field boundaries
- If ASSIGN fails, field-symbol is set to „unassigned“
- New ... RANGE addition allows the permissible boundaries to be expanded

- In the past, ASSIGN with offset/length access exceeding field boundaries has been used in some cases for processing "repetition groups" within structures. Now, there is a new statement „ASSIGN ... INCREMENT..." available for processing „repetition groups" within structures on a more abstract and a more secure level.

Class **CL_ABAP_CHAR_UTILITIES**

■ Constant attributes with system specific values

<code>charsize</code>	length of 1 character in bytes
<code>newline</code>	
<code>cr_lf</code>	
<code>form_feed</code>	
<code>horizontal_tab</code>	
<code>vertical_tab</code>	
<code>backspace</code>	
<code>minchar</code>	X'00' in non-Unicode systems, U+0000 in Unicode systems
<code>maxchar</code>	X'FF' in non-Unicode systems, U+FFFD in Unicode systems

Example

```
CLASS cl_abap_char_utilities DEFINITION LOAD.  
DATA: text TYPE string.  
REPLACE cl_abap_char_utilites=>horizontal_tab  
      WITH space INTO text.
```

- Use the constant attributes rather than hard coding byte values

Reading / Writing Different Text Formats

```
OPEN DATASET dsn IN TEXT MODE
      ENCODING (DEFAULT | UTF-8 | NON-UNICODE).
TRANSFER text TO dsn.
READ DATASET dsn INTO text.
```

- Only character-like fields allowed for reading / writing text files
- Explicit open required in Unicode enabled programs

Reading / Writing Legacy Formats

```
OPEN DATASET dsn IN LEGACY (TEXT | BINARY) MODE
      ... (LITTLE | BIG) ENDIAN
      ... CODEPAGE cp.
```

- Reading or writing data in a format compatible to non-Unicode systems
- Not character-like structures allowed

- If you open your text file with ENCODING DEFAULT, the text format used for reading or writing data will be platform dependent:
 - ◆ In a non-Unicode system, data will be written in non-Unicode format
 - ◆ In a Unicode-system, data will be written in Unicode format UTF-8

Conversion classes

- Code page conversion
 - ◆ Unicode / non-Unicode code pages
- Endian conversion
 - ◆ little endian / big endian byte order
- Character conversion
 - ◆ Unicode codepoint / ABAP character

ABAP Class	Conversion
CL_ABAP_CONV_IN_CE	any code page → system code page
CL_ABAP_CONV_OUT_CE	system code page → any code page
CL_ABAP_CONV_X2X_CE	any code page → any code page

- Use this classes to replace TRANSLATE CODEPAGE (and TRANSLATE NUMBER FORMAT)

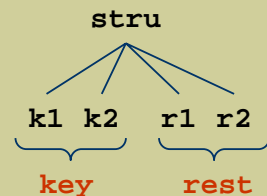
■ New ABAP Features – Includes with Group Names

Symbolic Access to Includes of Structures

```
TYPES: BEGIN OF t_key,  
      k1(2) TYPE x,  
      k2(2) TYPE c,  
      END OF t_key.
```

```
TYPES: BEGIN OF t_rest,  
      r1(10) TYPE c,  
      r2(10) TYPE c,  
      END OF t_rest.
```

```
DATA: BEGIN OF stru.  
      INCLUDE TYPE t_key as key.  
      INCLUDE TYPE t_rest as rest.  
DATA: END OF stru.  
DATA: skey TYPE t_key,  
      srest TYPE t_rest.
```



Pre-Unicode

```
skey = stru(4).  
srest = stru+4(20).  
WRITE: stru-r2.
```

Unicode enabled with group names

```
skey = stru-key.  
srest = stru-rest.  
WRITE: stru-r2.
```

- Use this feature to avoid offset programming
- The example above shows the use of includes with group names defined in ABAP. Similarly, you can introduce group names for includes in DDIC structures (transaction SE11 → tab strip components → entry in column 'group')

Using fields of type xstring as data containers

■ Writing data to an xstring.

```
DATA: my_buffer TYPE xstring.  
      data1      TYPE some_type.  
  
...  
  
EXPORT id = data1 TO DATA BUFFER my_buffer.
```

- ◆ Data is stored in a platform-independent format
- ◆ Contents of xstring can be exchanged with any other 6.10-system (Unicode and non-Unicode)

■ Reading data from an xstring

```
FORM read_buffer USING buffer TYPE xstring.  
  DATA: fld2 TYPE some_type.  
  IMPORT id = fld2 FROM DATA BUFFER buffer.  
  
  ...  
  
ENDFORM.
```

- ◆ Automatic conversion of data during import

- Use this feature to avoid using big character fields as container

Creating Data Objects Dynamically

- Creating and accessing data objects on the heap

```
DATA: dref TYPE REF TO data.  
CREATE DATA dref TYPE sometype.  
CREATE DATA dref TYPE (typename).  
CREATE DATA dref TYPE c LENGTH len.  
CREATE DATA dref TYPE STANDARD TABLE OF (typename)  
ASSIGN dref->* TO <f>.           "access data object
```

Casting to User Defined Types

- Look at the contents of a field as a value of another type

```
FIELD-SYMBOLS: <f> TYPE any.  
ASSIGN fld TO <f> CASTING TYPE sometype.  
ASSIGN fld TO <f> CASTING TYPE (typename).
```

- fld must provide sufficient alignment and length for the given type

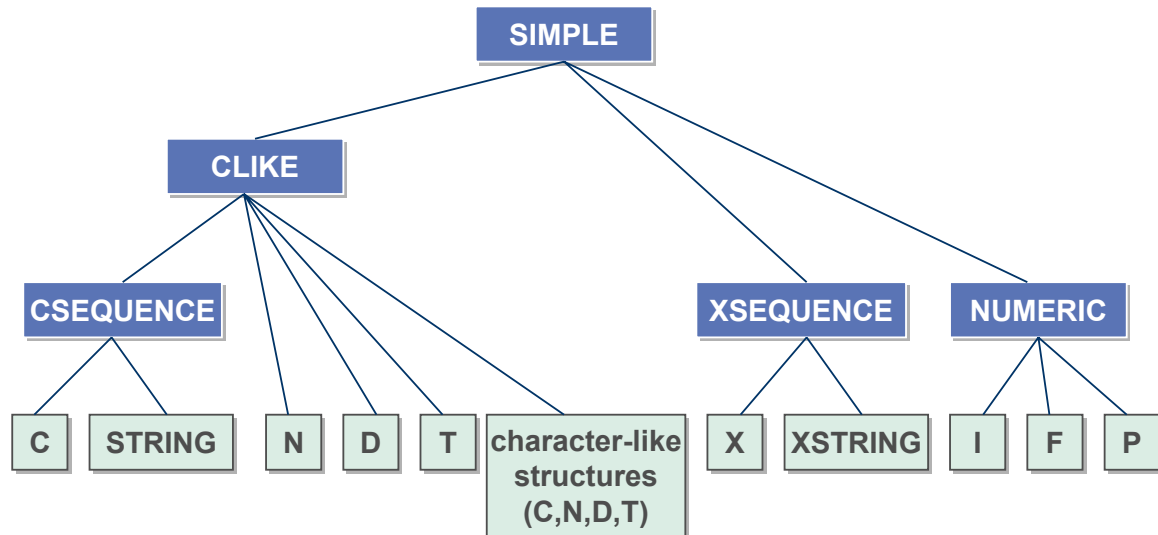
- Use this to create data objects with fitting type rather than using big character fields as container
- Example: Using dynamically created data objects as a work area for dynamic database operations

```
FORM write_column USING tname TYPE c  
                        cname TYPE c.  
  
DATA: dref TYPE REF TO data.  
FIELD-SYMBOLS: <wa> TYPE any,  
               <comp> TYPE any.  
  
CREATE DATA dref TYPE (tname).  
ASSIGN dref->* TO <wa>.  
  
SELECT * FROM (tname) INTO <wa>.  
  
ASSIGN COMPONENT cname OF STRUCTURE <wa> TO <comp>.  
IF sy-subrc <> 0. EXIT. ENDIF.  
WRITE: / <comp>.  
  
ENDSELECT.  
  
ENDFORM.
```

■ New ABAP Features - Generic Types

New generic types for parameters and field-symbols

- Eliminate untyped parameters or field-symbols for improved security and performance



New ABAP Features – Enhancement Categorization

If you are writing software for others you may have the following

Problem

- Enhancements on structures or tables may affect your coding:
 - ◆ Syntax-/runtime errors
 - ◆ Changed behavior (e.g. damaged or changed data)

Solution

- Maintaining the enhancement category in the DDIC: SE11 (Extras -> Enhancement Category)
 - ◆ Can not be enhanced
 - ◆ Can be enhanced - character like
 - ◆ Can be enhanced – character and numerical type
 - ◆ Can be arbitrarily enhanced
- Additional checks are done on your ABAP programs (SLIN) and show possible problems in allowed enhancement situations

- The feature is available as of basis release 6.20
- For details please look at note 493387

ABAP lists: Difference between memory and display length

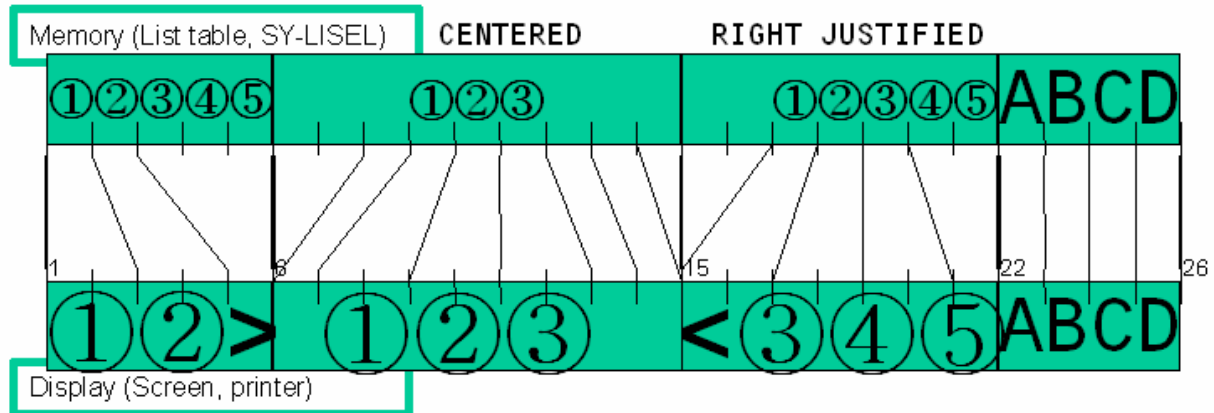
'한'	Character units in the memory	Display columns
Non-Unicode	2	2
Unicode	1	2

➔ **1 Character <> 1 Display Column**

- Unicode systems normally continue to use the old-fashioned non-proportional fonts on ABAP lists. Because of that nothing has changed on the output side and the former double byte characters are still twice as wide as European characters. But Unicode systems store the data in a different way: Also Asian characters can be stored in a character field of length 1.
- In fact using ABAP lists itself is old fashioned as they cannot use proportional fonts and lack a couple of other necessary features. Therefore the new ABAP List Viewer has replaced most ABAP lists on the screen. For formular printing on the other hand more powerful tools as Sapscrip or Smart forms should be used instead of ABAP list printing.

■ New ABAP Features – ABAP list programming

Handling for character fields:



Truncation may be done during display to synchronize memory length and display length at the field boundaries.



© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP

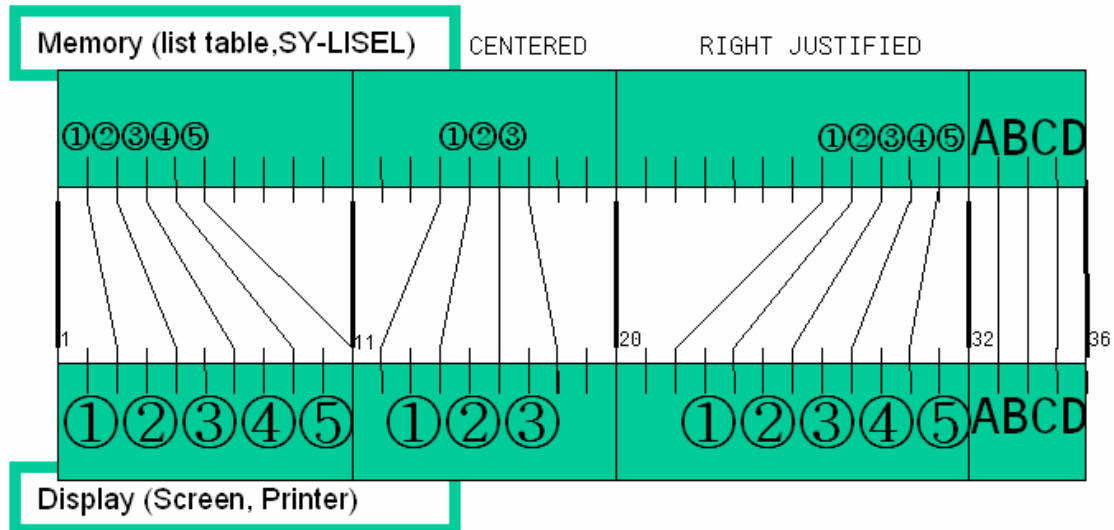
- The circled numbers are two columns wide at the display. The shown example results from the following data declarations and WRITE statements:

```
DATA F1(5). F1 = '①②③④⑤'.
DATA F2(3). F2 = '①②③'.
DATA F3(6). F3 = '①②③④⑤ '. "one space at the end
DATA F4(4). F4 = 'ABCD'.
```

```
WRITE: AT /1 F1 NO-GAP
      , AT (9) F2 CENTERD NO-GAP
      , AT (7) F3 RIGHT JUSTIFIED NO-GAP
      , F4.
```

■ New ABAP Features – ABAP list programming

Handling for strings:



Padding is done in the list table (here for the first field S1) to synchronize memory length and display length at the field boundaries.

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



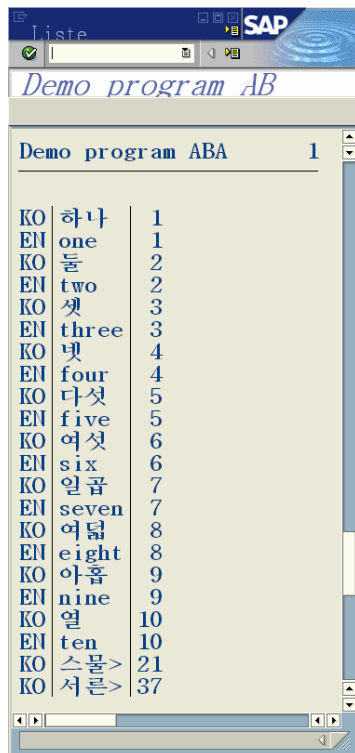
- The circled numbers are two columns wide at the display. The shown example results from the following data declarations and WRITE statements:

```
DATA S1 TYPE STRING. S1 = '①②③④⑤'.
DATA S2 TYPE STRING. S2 = '①②③'.
DATA S3 TYPE STRING. S3 = '①②③④⑤'.
DATA S4 TYPE STRING. S4 = 'ABCD'.
```

```
WRITE: AT /1 S1 NO-GAP
      , (9) S2 CENTERD NO-GAP
      , (12) S3 RIGHT JUSTIFIED NO-GAP
      , S4.
```

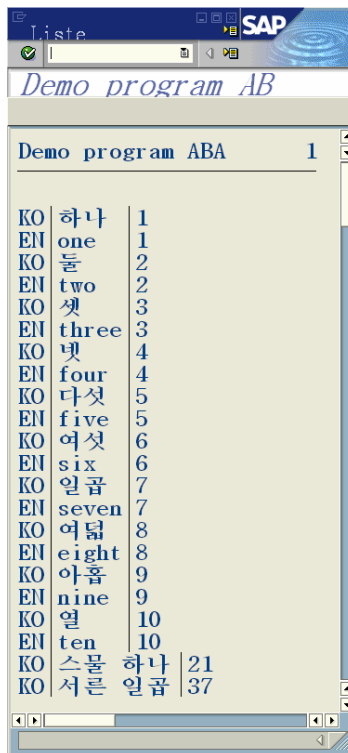
New ABAP Features – Different list types

Half width (Default)



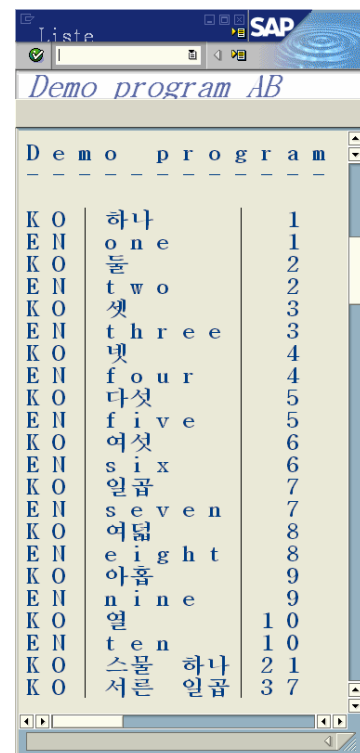
Demo program ABA 1		
KO	하나	1
EN	one	1
KO	둘	2
EN	two	2
KO	셋	3
EN	three	3
KO	넷	4
EN	four	4
KO	다섯	5
EN	five	5
KO	여섯	6
EN	six	6
KO	일곱	7
EN	seven	7
KO	여덟	8
EN	eight	8
KO	아홉	9
EN	nine	9
KO	열	10
EN	ten	10
KO	스물>	21
KO	서른>	37

Dynamic



Demo program ABA 1		
KO	하나	1
EN	one	1
KO	둘	2
EN	two	2
KO	셋	3
EN	three	3
KO	넷	4
EN	four	4
KO	다섯	5
EN	five	5
KO	여섯	6
EN	six	6
KO	일곱	7
EN	seven	7
KO	여덟	8
EN	eight	8
KO	아홉	9
EN	nine	9
KO	열	10
EN	ten	10
KO	스물 하나	21
KO	서른 일곱	37

Full width



Demo program ABA 1		
K O	하나	1
E N	one	1
K O	둘	2
E N	two	2
K O	셋	3
E N	three	3
K O	넷	4
E N	four	4
K O	다섯	5
E N	five	5
K O	여섯	6
E N	six	6
K O	일곱	7
E N	seven	7
K O	여덟	8
E N	eight	8
K O	아홉	9
E N	nine	9
K O	열	10
E N	ten	10
K O	스물 하나	21
K O	서른 일곱	37

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



- In order to give users the possibility to switch to a truncation-free output, such an option is offered in the menu System -> List -> Display type.

- There will be three different types:

Half-width (= DEFAULT):

output with truncation where necessary.

In this representation the layout is saved, but data truncation may occur.

Dynamic:

increase output length where necessary to output all data.

In this representation the layout may be destroyed, but all data is visible.

Full-width:

Output the list table content with the half-width letters spaced. In this representation the layout and the data is saved, but the list has an unusual look and feel

New ABAP statements

- SET/GET CURSOR *MEMORY* OFFSET
- Dynamic output length: WRITE (*) field.
Maximum output length: WRITE (**) field.

Utility Class CL_ABAP_LIST_UTILITIES

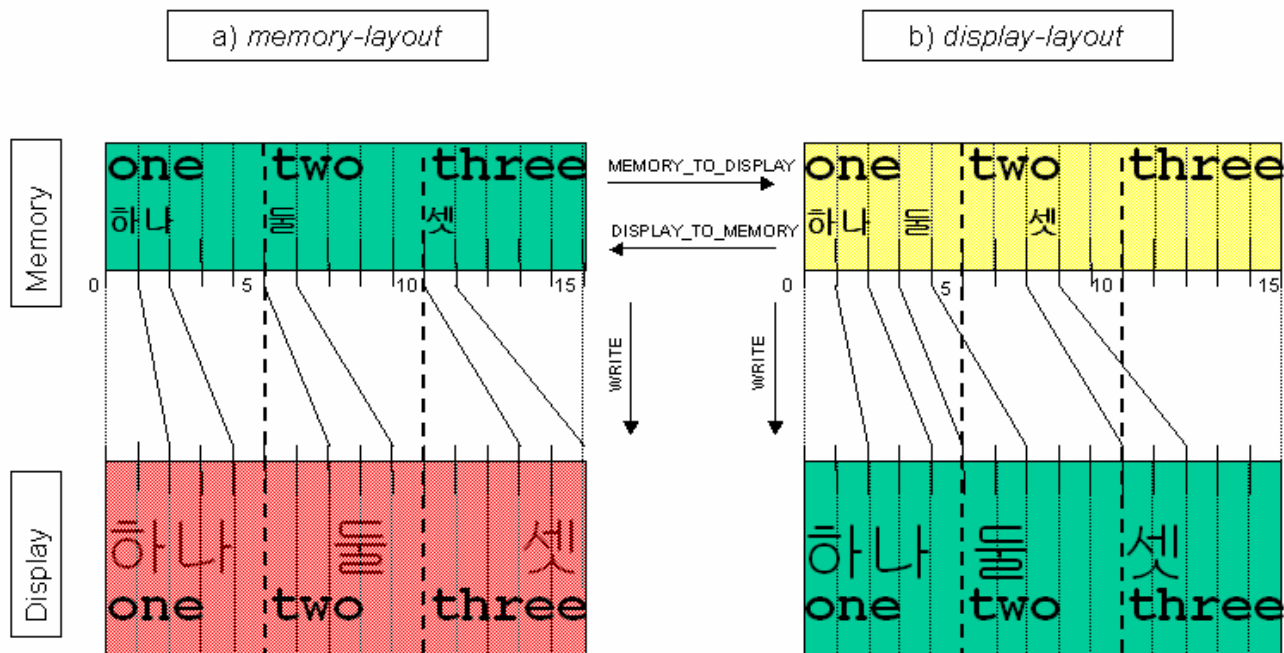
- Calculating display lengths
- Conversions display length \leftrightarrow memory length inside fields
- Handling of implicit field boundaries

See note 541299 for details

In detail the class CL_ABAP_LIST_UTILITIES contains:

- Calculating display lengths:
 - ◆ DEFINED_OUTPUT_LENGTH :
 - ◆ DYNAMIC_OUTPUT_LENGTH :
 - ◆ MAXIMUM_OUTPUT_LENGTH :
- Conversions display length \leftrightarrow memory length inside fields
 - ◆ DISPLAY_OFFSET / MEMORY_OFFSET
- Handling of implicit field boundaries
 - ◆ MEMORY_TO_DISPLAY / DISPLAY_TO_MEMORY
 - ◆ STRUCTURE_TO_DISPLAY / DISPLAY_TO_STRUCTURE
 - ◆ FRAME_SEPARATED_TO_DISPLAY / DISPLAY_TO_FRAME_SEPARATED

Field with implicit structure



© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



- Fields, that have a implicit sub structuring which is invisible to the ABAP list processor when writing the field as a whole, need a special treatment.
- Fields with implicit substructure appear in two different flavors: in *memory-layout* and *display-layout*:
 - ◆ A field in memory-layout is well suited for manipulations with memory offsets and lengths but must not be output to the display as it is, as the implicit layout is lost at the display.
 - ◆ The display oriented one is well suited for direct display output, but manipulations with memory offsets and lengths will destroy the implicit layout or even overwrite content.
- In order to convert a field with implicit sub structuring from the memory-layout into the display-layout (and vice versa) different types of methods are provided. The difference lays in the way that the field boundaries are determined. The most general method is:

CL_ABAP_LIST_UTILITIES=>MEMORY_TO_DISPLAY / DISPLAY_TO_MEMORY

The implicit field boundaries are explicitly supplied via an offset table.

Golden rules for ABAP list programming

- a) Don't mix up display length and memory length**
- b) Don't smudge field boundaries**
- c) Don't overwrite parts of fields**
- d) Don't do self programmed right-justified or centered**
- e) Don't do self programmed scrolling (memory based)**
- f) Don't forget to specify sufficient output length, if all data needs always to be visible**



Contents

Part I – SAPs approach to Unicode

- Demo – Unicode vs. Non-Unicode R3
- Unicode Essentials
- Transparent Unicode Enabling for R/3

Part II – Unicode Enabled ABAP

- Unicode Restrictions
- New ABAP Features

Part III – Tools for Unicode Enabling

- Migration to Unicode
- Unicode Scan UCCHECK
- Coverage Analyzer SCOV

Exercises

■ Migrating to Unicode Enabled ABAP

Step 1

- In **non-Unicode** system
- Adapt all ABAP programs to Unicode syntax and runtime restrictions
- Set attribute "**Unicode enabled**" for all programs

Step 2

- Set up a **Unicode** system
 - ◆ Unicode kernel + Unicode database
 - ◆ Only ABAP programs with the Unicode attribute are executable
- Do **runtime tests in Unicode system**
 - ◆ Check for runtime errors
 - ◆ Look for semantic errors
 - ◆ Check ABAP list layout with former double byte characters

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



- Step 1 can also be conducted in the Unicode system. However that's the brute force method, because at the beginning none of your programs is running. You make them work one by one doing the Unicode enablement and setting the Unicode attribute. Doing step 1 in a non-Unicode system allows for a smooth transition, because at the beginning all programs are running and you set the Unicode attribute one by one only after you made sure, that the program is Unicode compliant.
- In most cases, a program should behave identically in non-Unicode and Unicode systems after the "Unicode enabled" flag has been set. All of the runtime errors caused by the new restrictions for unicode-enabled programs can already be detected by tests in non-Unicode systems.
- Final tests however have to be done in the Unicode system, because some semantic differences will show up only in the Unicode system. For example the difference between DESCRIBE FIELD f LENGTH len IN BYTE MODE and DESCRIBE FIELD f LENGTH len IN CHARACTER MODE is visible only in the Unicode system.

■ Step 1 – Unicode Enabling with UCCHECK

Use UCCHECK to analyze your applications:

- Remove errors
- Inspect statically not analyzable places (optional)
 - ◆ Untyped field symbols
 - ◆ Offset with variable length
 - ◆ Generic access to database tables
- Set unicode program attribute using UCCHECK or SE38 / SE24 / ...
- Do additional checks with SLIN (e.g. matching of actual and formal parameters in function modules)

Transaction UCCHECK

Program Edit Goto System Help

Check a Program Set for Syntax Errors in Unicode Environment

Docu for ABAP + Unicode UCCHECK Documentation

Object Selection

Object name ZTECHED_UNICODE_E... to

Object Type to

Author (TADIR) to

Package to

Original system to

☒ Check only programs where the Unicode flag is unchecked

☒ Include only Objects with Object Repository Entry (TADIR)

☐ Exclude \$* Packages

Restriction of Program Set to Prevent Timeout

Maximum Number of Programs 50

Statements that cannot be analysed statically

☒ Display lines that cannot be analyzed statically

☐ Show also Locations Hidden with *#EC *

Includes to Be Displayed LSVIM* to

Application-specific Checks

☒ View Maintenance

☒ Obsolete Function Modules: UPLOAD/DOWNLOAD

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



- Essentially, the transaction UCCHECK does the ABAP syntax check for the selected programs as if the "Unicode enabling" attribute has already been set.
- If you check „Display lines that cannot be analyzed statically“ you will get hints on places that may have problems at runtime

UCCHECK – Setting Unicode Flag

SAP

System Help

SAP

Result of Unicode Syntax Check

Ex...	Program	Include	Row	Message
❌	ZTECHED_UNICODE_EXERCISE_4	ZTECHED_...	60	"ENTRY-CONTENTS" and "PERS" are not mutually convertible in a Unicode program.
❌	ZTECHED_UNICODE_EXERCISE_4	ZTECHED_...	84	"ENTRY-CONTENTS" and "PERS" are not mutually convertible in a Unicode program.
❌	ZTECHED_UNICODE_EXERCISE_4	ZTECHED_...	104	"PERS" and "CONTAINER_LINE-CONTENTS" are not mutually convertible in a Unicode program.
❌	ZTECHED_UNICODE_EXERCISE_4	ZTECHED_...	104	program.
⚠️	ZTECHED_UNICODE_EXERCISE_5	ZTECHED_...	42	The system could not perform a static compatibility check on the current statement, because of untyped or generic operands. The system will only carry out this check at runtime.
⚠️	ZTECHED_UNICODE_EXERCISE_5	ZTECHED_...	42	The system cannot perform a static check on the validity of the offset/ length entries for operand "CONTAINER+<DT>-OFFSET(<DT>-INTLEN)". They will be checked at runtime.
⚠️	ZTECHED_UNICODE_EXERCISE_5	ZTECHED_...	45	The system cannot perform a static check on the validity of the offset/ length entries for operand "CONTAINER+<DT>-OFFSET(<DT>-INTLEN)". They will be checked at runtime.
⚠️	ZTECHED_UNICODE_EXERCISE_5	ZTECHED_...	45	The system cannot perform a static check on the validity of the offset/ length entries for operand "CONTAINER+<DT>-OFFSET(<DT>-INTLEN)". They will be checked at runtime.
⚠️	ZTECHED_UNICODE_EXERCISE_5	ZTECHED_...	48	The system cannot perform a static check on the validity of the offset/ length entries for operand "CONTAINER+<DT>-OFFSET(<DT>-INTLEN)". They will be checked at runtime.
⚠️	ZTECHED_UNICODE_EXERCISE_5	ZTECHED_...	48	The system cannot perform a static check on the validity of the offset/ length entries for operand "CONTAINER+<DT>-OFFSET(<DT>-INTLEN)". They will be checked at runtime.
✅	ZTECHED_UNICODE_SOLUTION_1	ZTECHED_...	0	The system found no Unicode syntax errors
✅	ZTECHED_UNICODE_SOLUTION_2	ZTECHED_...	0	The system found no Unicode syntax errors
✅	ZTECHED_UNICODE_SOLUTION_3	ZTECHED_...	0	The system found no Unicode syntax errors
✅	ZTECHED_UNICODE_SOLUTION_4	ZTECHED_...	0	The system found no Unicode syntax errors

What to do with the places that can only be checked at runtime ?

- Reduce their number
 - ◆ In many cases you can specify the type of parameters and field-symbols
 - ◆ Use generic ABAP types where necessary
 - ◆ Mark those places that really need untyped parameters due to some kind of generic programming with “#EC * as OK after you did revise them.
- Do → Runtime tests

■ Step 2 – Testing Your Application

Final tests in the Unicode system

Runtime tests, Runtime tests, Runtime tests, ...

- ◆ Because the amount of warnings due to statically not analyzable places may be very large, you cannot type everything. In this case you have to rely on run-time tests.
- ◆ Some semantic problems may be seen only in the Unicode system (e.g. byte or character length)
- ◆ ABAP list layout can be checked only manually

Monitoring of runtime tests:

Having test plans is good, knowing the coverage of the test is better:

Use the *ABAP Coverage Analyzer* to monitor runtime tests

Coverage Analyzer (Transaction SCOV)

- persistently traces the execution of all program objects within one system
- traces all processing blocks
 - ◆ i.e. FORMS, Methods, Modules... and ABAP events
- collects Information
 - ◆ Number of calls
 - ◆ Number of runtime errors
 - ◆ Number of program changes

Two Different Target Groups

- Developers
 - ◆ Help to see in detail which parts of your programs are used and which are not
- Quality Managers
 - ◆ Determine the overall system coverage during a test phase

- You will find a complete description of the coverage analyzer at transaction SCOV → Menu Help → Application help
- A good overview is found in SAP Professional Journal 2002 Vol4/5:
“Improve Testing by Tracing ABAP Program Execution: Take a Closer Look with the Coverage Analyzer”

Coverage Analyzer – Details View for Developers

Exception Type	Name	Class	Acc. Exec.	Acc. Err.	Acc. Chng.	Curr. Exec.	Curr. Err.
ESEL			15	0	1	11	0
FORM	ENTER_VALUE		43	0	1	26	0
	F0		2	0	1	1	0
	F00		3	0	1	1	0
	F01		2	0	1	1	0
	F1		16	0	1	11	0
	F10		4	0	1	1	0
	F100		1	0	1	0	0
	F101		3	0	1	1	0
	F11		12	6	1	10	2
	START		18	0	1	12	0
PROG			18	0	1	12	0
SSEL			18	0	1	12	0

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP

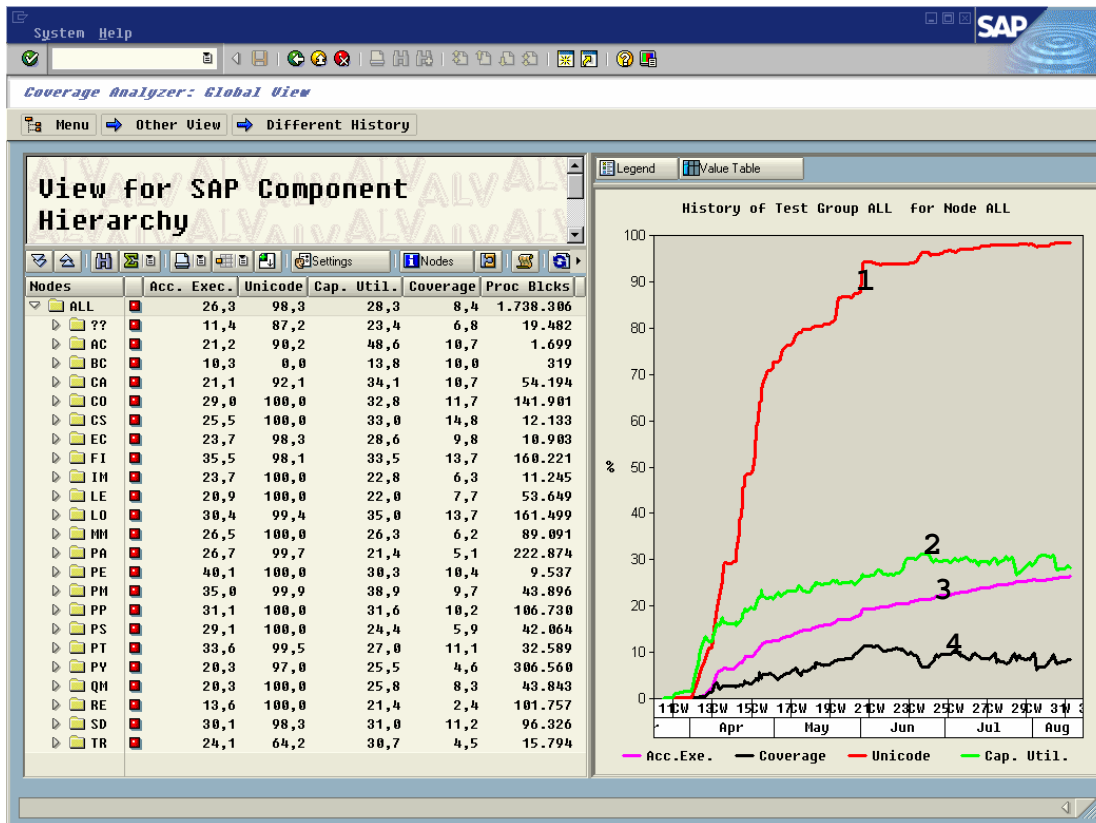


- In order to look at the execution details of your program, choose Coverage Analyzer → Display → Details, type in the name of your program and double click on the line with your program. You see a list of the processing blocks with the following column headers (see also F1 help):

- ◆ **Acc.Exec (Accumulated Executions) :**
This counter records the number of times that a processing block has been executed since the start of the Coverage Analyzer.
- ◆ **Acc.Err. (Accumulated Runtime Errors) :**
This counter records the number of times that a runtime error occurred during the execution of a processing block since the start of the Coverage Analyzer.
- ◆ **Acc.Chng. (Accumulated Changes) :**
This counter records the number of times a processing block is reset. A processing block is 'reset' whenever the program in which it is contained is explicitly or implicitly changed.
- ◆ **Curr.Exec. (Current Executions) :**
This counter records the number of times that a processing block has been executed in the current version (since the last reset).
- ◆ **Curr.Err. (Current Runtime Errors) :**
This counter records how often a runtime error occurred during the execution of a processing block in the current version (since the last reset) of the Coverage Analyzer

- In the example above, you can see that the subroutine F11 has been executed 12 times with a total of 6 runtime errors. The program (and thus F11) was changed once. After that change, F11 was called 10 times with 2 runtime errors.

Coverage Analyzer – Global View for QM



© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



- In order to get an overview of the complete system activities, choose Coverage Analyser → Display → Global. You can see a list of condensed results ordered by development class or author. In addition you can see the change of different values over time.
- In the figure above you see
 1. Unicode
This value indicates how many percent of the processing blocks have the Unicode flag set (the flag itself is set per program)
 2. Capacity Utilization
This value is computed as the ratio of used processing blocks to loaded processing blocks
 3. Accumulated Executions (Percent)
This value indicates in percent how many processing blocks have been executed since the start of the Coverage Analyzer.
 4. "Tested" processing blocks (Percent)
This value indicates in percent how many processing blocks have been executed in the actual version without runtime errors.
- In the example above you can see that currently 98% of the processing blocks in the system belong to a program that has the Unicode flag set, 26% have been executed since start of the coverage analyzer, 9% have been executed in the active version without errors and the capacity utilization is 29%.

Part I – SAPs approach to Unicode

- Demo – Unicode vs. Non-Unicode R3
- Unicode Essentials
- Transparent Unicode Enabling for R/3

Part II – Unicode Enabled ABAP

- Unicode Restrictions
- New ABAP Features

Part III – Tools for Unicode Enabling

- Migration to Unicode
- Unicode Scan UCCHECK
- Coverage Analyzer SCOV

Exercises

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



Technical information:

The exercises are based on the following Releases/Patchlevels

- Basis 6.20 Unicode system Support Package >= 21, Kernel Patch level >= 401
- Basis 6.20 non Unicode system Support Package >= 21, Kernel Patch level >= 401, Konfiguration as MDMP System (RSCPINST) with at least German, Japanese, Korean and English Locale installed
- SAPGUI 6.20 Patch Level >=34.
- Users were created with transaction BC_TOOLS_USER
- In order to show and experiment with multilingual data you may create a table with the fields COLOR, /SPRAS/NAME with the types CHAR 1 / LANG 1 / CHAR 6 and fill the table with report RSCPCOLORS . This may be done in both a non Unicode MDMP and a Unicode system. To display the content use SE16 → F6 → ALV Grid → Enter → F7 → F8.

■ Exercise I - Unicode Enabled ABAP

Using UCCHECK to remove static Unicode syntax errors

- Make your own copy of the programs
TECHED_UNICODE_EXERCISE_1/2/3/4 and 5 in the non-Unicode system.
- Inspect the programs using the transaction UCCHECK.
- Remove all static Unicode errors and set the Unicode attribute.

Find critical places with UCCHECK

- Make your own copy of program TECHED_UNICODE_EXERCISE_6 in the non-Unicode system.
- Run the program, check for static errors with UCCHECK and set the Unicode attribute.
- Try to rerun the program.
- Analyze the program with the UCCHECK option “statically not analyzable places”.
- Remove warnings by properly typing parameters and variables.
- Remove Unicode runtime errors.

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



Hints for the exercises:

- If not already existing copy TECHED_UNICODE_EXERCISE_n to ZTECHED_UNICODE_EXERCISE_n_XX where XX ist the two digit group number
- Solutions are in TECHED_UNICODE_SOLUTION_n
- Don't forget to activate the programs before checking with UCCHECK
- In the object selection uncheck the option “Exclude \$* Packages”
- TECHED_UNICODE_EXERCISE_6:
 - ◆ Use “#EC * to hide warnings from places that cannot be removed by proper typing.
 - ◆ Use generic types for the subroutine F00.

■ Exercise II – Unicode Enabled ABAP

Using SCOV to screen runtime tests

- Generate a copy of program TECHED_UNICODE_EXERCISE_7 on the Unicode system.
- Look in the Coverage Analyzer for the list of processing blocks.
- Execute the program several times and look in the Coverage Analyzer for the coverage of the different processing blocks.
- Find the errors in form F100 and F11.

Look at the ABAP list layout

- Generate a copy of the programs TECHED_UNICODE_EXERCISE_8/9/10 in the Unicode system.
- Execute the programs. Find and understand the pitfalls of the shown list programming techniques.
- Remove the errors.

Logon in the Unicode system and do SCOV runtime tests:

- The coverage analyzer has already be switched on for you to activate data collection at SCOV → Administration → “On/Off, Status” → Switch Coverage Analyzer On/Off, Button “On” . Before the “Data Collection: Bckgrd Server” has been maintained at SCOV → Administration → Settings
- If you want to cover all subroutines, you should look at the program coding with the debugger. It is just a simple tree structure.
- If you don’t want to spend too much time, try the following number combinations
 - ◆ 0/0
 - ◆ 0/1
 - ◆ 1/0/1/0
 - ◆ 1/0/1/1
 - ◆ 1/1
- Finally, try the following path:
 - ◆ 1/0/42 (Unicode error, visible only in a Unicode system)
 - ◆ 1/13 (Data dependent error, not Unicode specific)



Questions?



Q&A

© SAP AG 2003, TechED_Basel / ABAP151, Christian Hansen / 0

THE BEST-RUN BUSINESSES RUN SAP



Further Information

➔ **Public Web:**

Technical information: <http://service.sap.com/Unicode@SAP>

Customer contact: mail globalization@sap.com

➔ **Consulting Contact**

Roy Wood, VP SAP NetWeaver Consulting Practice (r.wood@sap.com)

➔ **Related SAP Education Training Opportunities**

SAP Unicode Learning Maps:

www.service.sap.com/okp --> My Learning Maps --> Unicode

➔ **Related Workshops/Lectures at SAP TechEd 2003**

SM202 Globalization: Unicode@SAP



**Please complete your session evaluation and
drop it in the box on your way out.**

Thank You !

The SAP TechEd '03 Basel Team

- No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.
- Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.
- Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.
- IBM®, DB2®, DB2 Universal Database, OS/2®, Parallel Sysplex®, MVS/ESA, AIX®, S/390®, AS/400®, OS/390®, OS/400®, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere®, Netfinity®, Tivoli®, Informix and Informix® Dynamic Server™ are trademarks of IBM Corporation in USA and/or other countries.
- ORACLE® is a registered trademark of ORACLE Corporation.
- UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.
- Citrix®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc.
- HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- JAVA® is a registered trademark of Sun Microsystems, Inc.
- JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
- MarketSet and Enterprise Buyer are jointly owned trademarks of SAP AG and Commerce One.
- SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies.