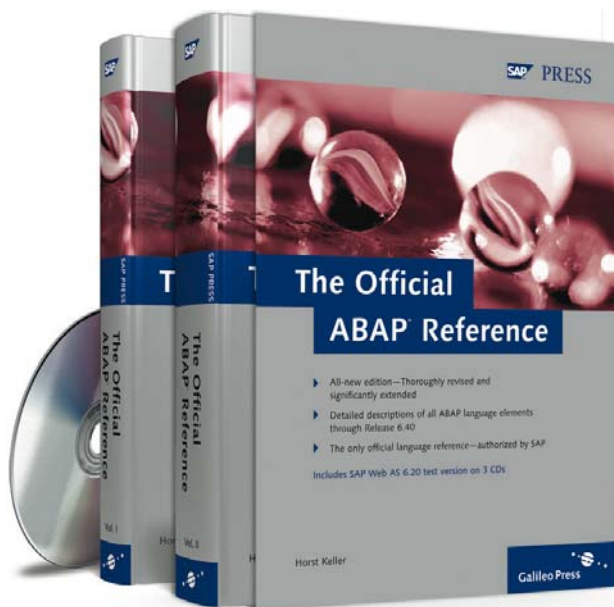


Horst Keller

The Official ABAP® Reference



Contents at a Glance

Volume I

Part 1	Syntax	33
Part 2	Program Structure	51
Part 3	Declarative Statements	113
Part 4	Generating Objects.....	207
Part 5	Calling and Exiting Program Units	227
Part 6	Program Flow Control	309
Part 7	Assignments	359
Part 8	Processing Internal Data	391

Volume II

Part 9	User Dialogs	529
Part 10	Processing External Data.....	741
Part 11	Program Parameters	893
Part 12	Program Processing	925
Part 13	External Programming Interfaces	959
Part 14	Obsolete Statements	1015
Part 15	Appendix	1089

Contents

Preface to the second edition	17
--------------------------------------	-----------

Preface to the first edition	19
-------------------------------------	-----------

1 Introduction and Overview	21
------------------------------------	-----------

1.1 Purpose and Objective	21
1.2 Described Releases	21
1.3 From SAP Basis to SAP Web Application Server	22
1.4 ABAP and Unicode	23
1.5 Book Structure	25
1.6 Search Options	28
1.7 Syntax Diagrams	29
1.8 Changes to the First Edition	30

Part 1 Syntax

2 ABAP Syntax	35
----------------------	-----------

2.1 ABAP Statements	35
2.2 ABAP Language Element	35
2.2.1 Operators	35
2.2.2 Operands	36
2.2.3 Names for Operands	37
2.2.4 Specifying Operands Statically or Dynamically	43
2.2.5 Data Objects in Operand Positions	43
2.3 Naming Conventions	47
2.4 Chained Statements	48
2.5 Comments	49

Part 2 Program Structure

3 Introductory Program Statements	53
--	-----------

3.1 Overview	53
--------------------	----

3.2	Executable Programs	54
3.2.1	Additions for the Basic List of the Program	55
3.2.2	Addition for the Message Class	57
3.2.3	Addition for Defining a Logical Database	57
3.3	Module Pools and Subroutine Pools	57
3.4	Function Groups	58
3.5	Class Pools	59
3.6	Interface Pools	59
3.7	Type Groups	59

4 Modularization Statements 61

4.1	Overview	61
4.2	Procedures	62
4.2.1	Methods	62
4.2.2	Function Modules	63
4.2.3	Subroutines	65
4.3	Dialog Modules	69
4.4	Event Blocks	70
4.4.1	Program Constructor	71
4.4.2	Reporting Events	72
4.4.3	Selection Screen Events	77
4.4.4	List Events	82
4.5	Source Code Modularization	88
4.5.1	Include Programs	89
4.5.2	Macros	90

5 Built-in Types, Data Objects, and Functions 93

5.1	Overview	93
5.2	Predefined Data Types	93
5.2.1	Predefined ABAP Types	93
5.2.2	Generic ABAP Types	97
5.2.3	Predefined Types in the ABAP Dictionary	98
5.2.4	The Built-In Data Type "Cursor"	102
5.3	Built-In Data Objects	102
5.3.1	The Space Constant	102
5.3.2	The Self-Reference me	102
5.3.3	ABAP System Fields	102
5.3.4	The Screen Structure	108
5.4	Predefined Functions	108
5.4.1	Mathematical Functions	108
5.4.2	Description Functions	111

Part 3 Declarative Statements

6 Declarative Statements for Data Types and Data Objects 115

6.1	Overview	115
6.1.1	Validity and Visibility	115
6.1.2	Data Types	116
6.1.3	Data Objects	116
6.1.4	Absolute Type Names	118
6.1.5	Statements for Declaring Data Types and Data Objects	120
6.2	Including Type Groups	121
6.3	Defining Data Types	122
6.3.1	Type Definitions with Built-In ABAP Types	123
6.3.2	Type Definition by Reference to Existing Types	124
6.3.3	Defining Reference Types	124
6.3.4	Defining Structured Types	126
6.3.5	Defining Table Types	127
6.3.6	Defining a Range Table Type	130
6.4	Declaring Variables	131
6.4.1	Additions for Data Objects	133
6.4.2	Elementary Data Objects of Built-In ABAP Types	134
6.4.3	Data Objects of Existing Types	135
6.4.4	Declaring Reference Variables	136
6.4.5	Declaring Structures	137
6.4.6	Declaring Internal Tables	138
6.4.7	Defining a Ranges Table	140
6.5	Declaring Static Attributes of Classes	141
6.6	Declaring Constant Data Objects	142
6.7	Declaring Static Data Objects in Procedures	143
6.8	Copying Structure Components	144
6.9	Declaring the Table Work Areas	146
6.9.1	Flat Table Work Area	146
6.9.2	Any Table Work Area	147
6.10	Declaring Field Symbols	150
6.10.1	Typing Field Symbols	150
6.10.2	Casting a Structure	151
6.11	Declaring an Extract Dataset	152

7 Defining Classes and Interfaces 153

7.1	Overview	153
7.2	Defining Classes	154
7.2.1	Declaration Section	154

7.2.2	Implementation Section	163
7.2.3	Declaring Classes	165
7.3	Defining Interfaces	166
7.3.1	Declaration	167
7.3.2	Declaring Interfaces	168
7.4	Declaring Components in Classes and Interfaces	169
7.4.1	Methods	169
7.4.2	Events	189
7.4.3	Implementing and Integrating Interfaces	192

8 Typing 199

8.1	Overview	199
8.2	Syntax of Typing	199
8.2.1	Generic Typing	200
8.2.2	Complete Typing	200
8.3	Checking the Typing	201
8.3.1	General Rules	202
8.3.2	Literals as Actual Parameters	202

Part 4 Generating Objects

9 Generating Objects and Data Objects 209

9.1	Overview	209
9.2	Generating Data Objects	209
9.2.1	Defining the Data Type Implicitly	210
9.2.2	Defining the Data Type Through Built-In ABAP Types	211
9.2.3	Defining the Data Type Through an Existing Type	213
9.2.4	Generating Reference Variables	215
9.2.5	Generating Internal Tables	216
9.2.6	Defining The Data Type Through Type Objects	217
9.2.7	Treatable Exceptions When Generating Data Objects	219
9.3	Generating Objects in ABAP Objects	219
9.3.1	Defining the Class Implicitly	220
9.3.2	Defining the Class Explicitly	221
9.3.3	Static Parameter Transfer	222
9.3.4	Dynamic Parameter Passing	222
9.3.5	Setting Of Object In Shared Memory	224
9.3.6	Treatable Exceptions When Generating Objects	225

Part 5 Calling and Exiting Program Units

10	Calling ABAP Programs	229
10.1	Overview	229
10.2	Calling Executable Programs	230
10.2.1	Flow of the Program Call	231
10.2.2	Additions for the Selection Screen	233
10.2.3	Additions for the Basic List	241
10.2.4	Additions for Background Processing	246
10.3	Calling Transactions	248
10.3.1	Calling a Transaction and Returning to the Calling Program	248
10.3.2	Calling a Transaction Without Returning to the Calling Program	255
11	Calling Processing Blocks	259
11.1	Overview	259
11.2	Calling Procedures	260
11.2.1	Overview	260
11.2.2	Method Calls	262
11.2.3	Calling Function Modules	274
11.2.4	Subroutine Call	284
11.3	Calling Event Handlers	290
11.3.1	Triggering Events	290
11.3.2	Registering Event Handlers	291
11.4	Calling Event Blocks	295
11.4.1	Triggering Events of a Logical Database	295
11.4.2	Triggering List Events	297
12	Exiting Program Units	299
12.1	Overview	299
12.2	Leaving Programs	299
12.3	Exiting Processing Blocks	300
12.3.1	Procedure for Exiting Processing Blocks	300
12.3.2	Exiting Processing Blocks with RETURN	301
12.3.3	Exiting Processing Blocks with EXIT	302
12.3.4	Exiting Processing Blocks with CHECK	302
12.3.5	Exiting GET Processing Blocks with REJECT	304
12.3.6	Exiting Processing Blocks of Executable Programs with STOP	305
12.4	Exiting Loops	306
12.4.1	Exiting Loops with EXIT	306

12.4.2	Exiting a Loop Pass with CONTINUE	307
12.4.3	Exiting a Loop Pass with CHECK	307

Part 6 Program Flow Control

13 Logical Expressions 311

13.1	Overview	311
13.2	Logical Expressions with Relational Operators	311
13.2.1	Relational Operators for All Data Types	312
13.2.2	Comparison Operators for Character-Type Operands	316
13.2.3	Relational Operators for Byte-Type Data Types	318
13.2.4	Relational Operators for Bit Patterns	319
13.3	Determining the Interval Range	320
13.4	Checking States	321
13.4.1	Checking the Assignment to a Field Symbol	321
13.4.2	Checking the Validity of a Reference	321
13.4.3	Checking an Operand for the Initial Value	322
13.4.4	Checking Output Parameters for Actual Parameters	323
13.4.5	Checking Formal Parameters for Actual Parameters	324
13.5	Evaluating a Selection Table	325
13.6	Boolean Operators and Parenthetical Expressions	327
13.6.1	Linking Logical Expressions with AND	328
13.6.2	Linking Logical Expressions with OR	328
13.6.3	Negating a Logical Expression with NOT	328
13.6.4	Parenthetical Expressions	329

14 Control Structures 331

14.1	Overview	331
14.2	Branching	331
14.2.1	Branching with IF	332
14.2.2	Branching with CASE	333
14.3	Loops	334
14.3.1	Unconditional Loops with DO	334
14.3.2	Conditional Loops with WHILE	337
14.4	Program Interruption	338

15 Exception Handling 341

15.1	Overview	341
15.1.1	Treatable Exceptions Prior to Release 6.10	341
15.1.2	Treatable Exceptions in Release 6.10 and Later	341

15.1.3	System Behavior After a Class-Based Exception	347
15.1.4	Interaction of Non-Class-Based and Class-Based Interactions	348
15.1.5	Statements for Exceptions	349
15.2	Class-Based Exceptions	349
15.2.1	Triggering Class-Based Exceptions	349
15.2.2	Handling Class-Based Exceptions	350
15.3	Catchable Runtime Errors	354
15.3.1	Defining and Triggering Catchable Runtime Errors	354
15.3.2	Handling Catchable Runtime Errors	354
15.4	Non-Class-Based Exceptions	356
15.4.1	Defining Non-Class-Based Exceptions	356
15.4.2	Triggering Non-Class-Based Exceptions	357
15.4.3	Handling Non-Class-Based Exceptions	358

Part 7 Assignments

16 Value Assignments 361

16.1	Overview	361
16.2	Assigning Data Objects	362
16.2.1	Assignment with Keyword	362
16.2.2	Assignment with an Assignment Operator	362
16.2.3	Multiple Assignments	363
16.3	Assigning Structure Components	364
16.4	Formatted Assignment	366
16.5	Converting A Packed Number	367

17 Setting References 369

17.1	Overview	369
17.2	Assigning Data Objects to Field Symbols	369
17.2.1	Specifying a Memory Area	371
17.2.2	Specifying the Data Type	377
17.2.3	Specifying the Range Limits	380
17.3	Initializing Field Symbols	383
17.4	Retrieving a Data Reference	384

18 Initializing Data Objects 387

18.1	Overview	387
18.2	Initializing Any Data Objects	387
18.2.1	Initializing with the Initial Value	387
18.2.2	Initializing with Other Contents	388

18.3	Initializing Internal Tables	389
18.4	Releasing Memory	390

Part 8 Processing Internal Data

19 Calculation Expressions 393

19.1	Overview	393
19.2	The COMPUTE Statement	393
19.3	Arithmetic Expressions	394
19.3.1	Arithmetic Operators	395
19.3.2	Compounding	395
19.3.3	Priority of Functional Methods	396
19.3.4	Calculation Type	396
19.3.5	Treatable Exceptions in Arithmetic Expressions	398
19.4	Bit Expressions	398
19.4.1	Bit Operators	399
19.4.2	Compounding	400

20 Calculation Statements 401

20.1	Overview	401
20.2	Addition	401
20.3	Subtraction	402
20.4	Multiplication	402
20.5	Division	403

21 Processing Byte and Character Strings 405

21.1	Overview	405
21.1.1	Byte Strings and Character Strings	405
21.1.2	Instructions for Byte and Character-String Processing	405
21.1.3	Operands in Byte and Character-String Processing	406
21.1.4	Handling Closing Blanks in Character-String Processing	407
21.2	Concatenating Byte and Character Strings	408
21.3	Searching Byte and Character Strings	409
21.3.1	Searching with FIND	409
21.3.2	Searching with SEARCH	412
21.4	Replacing Byte and Character Strings	416
21.4.1	Position-Based Replacement	417
21.4.2	Pattern-Based Replacement	418

21.5	Shifting Byte and Character Strings	420
21.5.1	Number of Places	421
21.5.2	Shifting Direction	423
21.5.3	Shifting Certain Characters Out of the Field	423
21.6	Splitting Byte and Character Strings	424
21.7	Condensing Character Strings	426
21.8	Converting Character Strings	427
21.9	Overlaying Character Strings	429
21.10	Converting Character Strings	430
21.10.1	Converting Upper/Lowercase	430
21.10.2	Pattern-Based Conversion	431
21.11	Setting and Reading Bits	432
21.11.1	Setting a Single Bit in a Byte String	432
21.11.2	Reading Single Bites from Byte Strings	433

22 Processing Internal Tables 435

22.1	Overview	435
22.2	Reading Internal Tables	436
22.2.1	Reading Single Lines	436
22.2.2	Loop Processing for Internal Tables	446
22.2.3	Control-Level Processing	451
22.2.4	Summation in Control Levels	453
22.3	Filling Internal Tables	455
22.3.1	Inserting Lines	455
22.3.2	Inserting Condensed Lines	459
22.3.3	Appending Lines	461
22.4	Processing Table Lines	465
22.4.1	Changing Lines	465
22.4.2	Deleting Lines	470
22.4.3	Sorting	476
22.5	Specifying Components	479
22.6	Searching Internal Tables	481
22.6.1	Search Options	482
22.7	Processing Special Internal Tables	483

23 Processing Extract Datasets 489

23.1	Overview	489
23.2	Defining the Line Structure	489
23.3	Filling the Extract Dataset	491
23.4	Sorting the Extract Dataset	492

23.5	Reading the Extract Dataset	494
23.6	Control-Level Processing	495

24 Attributes of Data Objects 499

24.1	Overview	499
24.2	Elementary Properties of Any Data Objects	499
24.2.1	Data Type and Number of Components	500
24.2.2	Length Used by the Data Object	502
24.2.3	Decimal Places	503
24.2.4	Output Length	503
24.2.5	Elementary Data Type in the ABAP Dictionary	504
24.2.6	Conversion Routines	505
24.3	Attributes of Internal Tables	506
24.3.1	Table Type	506
24.3.2	Number of Lines	507
24.3.3	Initial Memory Requirements	507
24.4	Distance Between Data Objects	508

Preface to the second edition

Barely two years have passed since the appearance of the first edition of the ABAP Reference. This new, second edition reflects the complete scope of ABAP elements produced for the current Release 6.40 and corrects and completely revises large sections of the previous text (see Section 1.8).

The most important innovations of Release 6.40 are the evolution of RTTI to RTTC and shared objects. In addition to the existing Run Time Type Information (RTTI), the Run Time Type Services (RTTS) now contain Run Time Type Creation (RTTC), which enables you to create and use any data types during the program runtime. This means that many ABAP developers have finally got what they wanted: They can create dynamic structures. The example in Section 9.2.6 shows, how this works. Shared objects are objects in the shared memory, which all programs of an application server can access. The related language elements and concepts are described in this book. Further important innovations in Release 6.40 are the introduction of simple transformations to serialize ABAP data to XML and vice versa, and the ABAP unit, which is a testing tool integrated in the language.

The SAP Web Application Server and the ABAP language has been a component of SAP NetWeaver since Release 6.30. SAP NetWeaver is an open integration and application platform which replaces and expands the preceding mySAP technology. In addition to the familiar ABAP environment, SAP Web Application Server—since Release 6.30—also contains an environment for the application development in Java. As this is not directly connected to ABAP language elements, this book will not explore the Java component of the Web Application Server.

As Mr Joachim Jacobitz, my co-author for the first edition, has since assumed different responsibilities within SAP, I am now the sole author of this book. In spite of this, however, Mr Jacobitz proofread each chapter with great enthusiasm and gave me many useful tips, for which I would like to sincerely thank him.

As the sole author of these voluminous two books, it is important for me to point out that this task would not have been possible without the "NetWeaver Developer Tools ABAP" group whose outstanding work I would like to emphasize at this point. As was the case in the first edition, the second edition was made possible by the constructive criticism and proofreading of many colleagues. In addition to the colleagues

mentioned in the foreword to the first edition, the following people definitely deserve a mention in the second edition: Ulrich Brink, Christian Hansen, Ulrich Koch, Jürgen Lehmann, Mathias Müller, Helmut Prestel, Michael Redford, Christian Stork, Wolf Hagen Thümmel and Christoph Wedler. I would like to express my sincere gratitude to all other colleagues, who have not been named here, but whom I also bothered with queries or who gave me tips in one way or another.

On this occasion I would like to express my most sincere gratitude to my student assistants, Ms Agnieszka Chelminska, Ms Beata Kouchnir and Mr Christian Pretzsch, who enthusiastically and actively assisted me in creating and correcting the manuscript and the revision of the online documentation.

I would like to thank the involved people at Galileo Press and my editor Mr Florian Zimniak, for the fact that they enabled the rapid publication of the revised version of a book which I consider very important, and for their continuous pleasant cooperation.

For the English edition I would especially like to thank John Parker from UCG, Stefan Proksch from Galileo Press, as well as the students Kathrin Sturmhöfel, Marc Langen, and Dariusz Chelminski for their commitment during the technical editing of the text.

I owe special thanks to Ute for her patience and understanding for the fact that even after the appearance of the first edition, I continued to write this book.

Walldorf, Germany, May 2004
Horst Keller

Preface to the first edition

The introduction of ABAP Objects in Release 4.6 represented a significant step towards contemporary programming techniques for the ABAP programming language. Still, the next major challenge in the language's future development was already looming: to meet the demands of an international, Internet-driven programming environment, Web Application Server—as successor to the SAP Basis Module—had to be made Unicode-capable, and with it the ABAP programming language. This requirement was met in Release 6.10.

However, implementing Release 6.10 or 6.20 does not mean that you will be forced to modify your programs, as long as you do not convert the SAP system to Unicode. Nonetheless, regardless of whether a program will be used in a Unicode or non-Unicode system, the new features implemented together with Unicode represent an important step towards more stable, bug-free programming.

Accordingly, the introduction of Unicode was a particular impetus for writing this book. The online ABAP keyword documentation in the system grew with the language as new terms were added and existing terms remained. It has a section on "ABAP Objects" and—since Release 6.10—a section on "Unicode" as well, while the instructions for traditional reporting are still written as if every ABAP program were a report that is linked with a logical database. Because this type of documentation is acceptable for context-sensitive help—but no longer sufficient for full, integrated documentation—we resolved to write an ABAP Reference work to follow our introduction to SAP programming.

The goals we defined were to completely revise and standardize the ABAP documentation. This not only involved changing the outlines and summaries of the individual statements, but also their content and style. To this effect, we have introduced a new form of illustration, "pseudo syntax," to break down any statement from its basic form to its individual components. This book also marks the first time that the requirements for all operands that can appear in ABAP statements have been described exhaustively and uniformly in one go. Our intent is to give you a comprehensive description of all the ABAP language elements that are active in the current Release 6.20. This includes detailed descriptions of language elements that have been flagged as obsolete, but which are only prohibited in the context of ABAP Objects. This ABAP Reference will help you to analyze any existing ABAP program, as well as write your own. While it is not intended to be a "Best Practices" book, we hope that giving you

precise descriptions of each statement will make their intent and purpose clearer than was previously the case.

Documenting a language boasting some 500 major language elements—while maintaining the same high quality and degree of detail throughout—was certainly no easy task, particularly since ABAP has been continually enhanced over the past 20 years and no element, once introduced, has ever been deleted (to ensure downward compatibility). This fact also explains the repeated delays in publishing this book, and we would like to thank all of you for waiting so patiently for its release. To compensate, the book you now hold in your hands documents the latest release of the ABAP language, and will remain valid for a long time to come. Despite our careful scrutiny of each and every chapter, we cannot discount the possibility that the book may still contain a few minor inconsistencies, and would be grateful to you for pointing out anything you might find. You can reach us at the SAP PRESS Web site under www.sap-press.com.

It would not have been possible to write this book in its current form without the direct and indirect help we enjoyed from a number of people during the authoring and proofreading processes. We would like to expressly thank Masoud Aghadavoodi, Thomas Bareiss, Adrian Goerler, Christian Jendel, Gerd Kluger, Björn Mielenhausen, Andreas Simon Schmitt and Christoph Stöck for their help. We thank Erhardt Vortanz for his assistance in writing the manuscript. We especially thank Andreas Blumenthal, Development Manager of the Business Programming Languages Group, for making this project possible in the first place and for more or less letting us decide what form it would take. We have the tireless Michael Demuth to thank for the fact that this book comes with an SAP system, this time as two CDs with SAP Web Application Server 6.10. Lastly, we would like to thank the staff of Galileo Press—particularly Iris Warkus and Florian Zimniak—for their help in proofreading the manuscript and for never giving up hope that we really would be finished at some point.

Horst Keller particularly thanks his wife Ute, who in the last few months nearly always found him hunched over his laptop, for her patience and willingness to sacrifice more of their already scarce free time together for yet another book project.

We hope you enjoy reading this ABAP Reference.

Walldorf, Germany, July 2002

Horst Keller and Joachim Jacobitz

Contents

Part 9 User Dialogs

25 Dynpros 531

25.1	Overview	531
25.1.1	User Interface	531
25.1.2	Screen Layout	532
25.1.3	Dynpro Fields	533
25.1.4	Dynpro Flow and Dynpro Sequences	534
25.2	Statements in the Dynpro Flow Logic	535
25.2.1	Dynpro Flow Logic Event Blocks	536
25.2.2	Calling Dialog Modules	537
25.2.3	Controlling Data Transfer and Flow Logic	541
25.2.4	Processing Chains	547
25.2.5	Processing Table Controls	549
25.2.6	Calling Subscreens	553
25.3	ABAP Statements for Dynpros	556
25.3.1	Calling a Dynpro Sequence	557
25.3.2	Setting the GUI Status	558
25.3.3	Determining the GUI Status	560
25.3.4	Setting the GUI Title	561
25.3.5	Suppressing the Display	563
25.3.6	Determining Attributes of Screen Elements	564
25.3.7	Modifying Attributes of Screen Elements	566
25.3.8	Setting the Cursor	567
25.3.9	Evaluating the Cursor Position	569
25.3.10	Declaring a Control	570
25.3.11	Initializing Table Control	576
25.3.12	Exiting Step-Loop Processing	576
25.3.13	Holding Input Data	577
25.3.14	Setting the Next Dynpro	578
25.3.15	Leaving a Dynpro	578

26 Selection Screens 581

26.1	Overview	581
26.1.1	Selection Screens as Dynpros	581
26.1.2	Selection Screen Tasks	581
26.1.3	GUI Status of Selection Screens	581
26.1.4	Selection Screen Events	582

26.1.5	Selection Screens and Logical Databases	582
26.1.6	Statements for Selection Screens	582
26.2	Creating and Laying Out Selection Screens	582
26.2.1	Creating Selection Screens	583
26.2.2	Laying Out Selection Screens	586
26.2.3	Adopting Elements of Other Selection Screens	601
26.2.4	Variants and Additions for Selection Screens of Logical Databases	606
26.3	Defining Parameters	610
26.3.1	Data Type of the Parameter	611
26.3.2	Attributes of Screen Elements	614
26.3.3	Attributes of the Value and the Passing of Values	620
26.3.4	Additions for Selection Screens of Logical Databases	623
26.4	Defining Selection Criteria	626
26.4.1	Data Type of the LOW and HIGH Selection Table Columns	629
26.4.2	Attributes of Screen Elements	631
26.4.3	Attributes of the Value and Passing of Values	634
26.4.4	Additions for Selection Screens of Logical Databases	636
26.5	Calling Selection Screens	638
26.5.1	Call via SUBMIT	638
26.5.2	Call via Report Transaction	638
26.5.3	Call via Dialog Transaction	639
26.5.4	Calling in the Program	639
26.5.5	Selection-Screen Processing	640

27 Lists 643

27.1	Overview	643
27.1.1	Lists as Screens	643
27.1.2	Lists in ABAP Programs	643
27.1.3	Basic List	643
27.1.4	Details Lists	643
27.1.5	Structure of a List	644
27.1.6	Print Lists	645
27.1.7	Lists and ABAP Objects	645
27.1.8	Lists and Unicode	645
27.1.9	Statements for List Processing	646
27.2	Creating Lists	647
27.2.1	Writing Data in Lists	647
27.2.2	Creating Horizontal Lines	673
27.2.3	Formatting Lists Section by Section	674
27.2.4	Displaying Blank Lines	681
27.2.5	Vertical Positioning of the List Cursor	681
27.2.6	Horizontal Positioning of the List Cursor	686
27.2.7	Fixed Area When Scrolling Horizontally	687
27.2.8	Page Breaks and Print Lists	688
27.2.9	Conditional Page Break	696
27.2.10	Saving Variables with List Lines	697

27.2.11	Print List Page Margins	698
27.2.12	Controlling Print Lists	699
27.3	Processing Lists in the List Buffer	703
27.3.1	Reading List Lines	703
27.3.2	Modifying List Lines	705
27.3.3	Scrolling Lists	708
27.3.4	Reading List Attributes	711
27.4	Evaluating the Displayed List at the Cursor Position	714
27.5	Display Attributes of Screen Lists	716
27.5.1	GUI Status of a Screen List	716
27.5.2	Title of a Screen List	718
27.5.3	Setting the Cursor	718
27.5.4	Lists in a Dialog Box	721
27.6	Calling and Exiting List Displays	722
27.6.1	Calling the Basic List Display	723
27.6.2	Leaving the List Display	724

28 Messages 727

28.1	Overview	727
28.1.1	Storing Messages	727
28.1.2	Message Types	727
28.2	Sending Messages	731
28.2.1	Specifying a Message	731
28.2.2	Specifying any Text	735
28.2.3	Additions to MESSAGE	736

Part 10 Processing External Data

29 Open SQL 743

29.1	Overview	743
29.1.1	Scope of Open SQL	743
29.1.2	Database Interface	743
29.1.3	Database Access	743
29.1.4	Client Handling	743
29.1.5	SAP Buffering	743
29.1.6	LUW	744
29.1.7	Open SQL Statements	744
29.2	Reading Data from Database Tables	745
29.2.1	Determining the Structure of the Result Set	747
29.2.2	Specifying Database Tables to Be Read	754
29.2.3	Specifying the Target Area	760
29.2.4	Restricting the Result Set	765
29.2.5	Combining Lines	777
29.2.6	Restricting Combined Lines	778

29.2.7	Sorting Result Set Lines	779
29.3	Reading Data from Database Tables Using a Cursor	782
29.3.1	Opening a Cursor	782
29.3.2	Reading Data Using a Cursor	783
29.3.3	Close Cursor	784
29.4	Inserting Data into Database Tables	786
29.4.1	Specifying the Database Table	787
29.4.2	Specifying the Source	788
29.5	Changing Data in Database Tables	790
29.5.1	Specifying the Database Table	790
29.5.2	Specifying Changes	791
29.6	Inserting or Changing Data in Database Tables	797
29.6.1	Specifying the Database Table	797
29.6.2	Specifying the Source	798
29.7	Deleting Data in Database Tables	799
29.7.1	Specifying the Database Table	800
29.7.2	Specifying the Lines	800
29.8	Work Areas in Open SQL Statements	803
29.9	Treatable Exceptions in Open SQL Statements	804

30 Native SQL 805

30.1	Overview	805
30.2	Including Native SQL	805
30.2.1	Host Variables	807
30.2.2	Cursor Processing	808
30.2.3	Calling Database Procedures	809
30.2.4	Defining the Database Connection	811
30.2.5	Implicit Cursor Processing	814
30.3	Leaving Native SQL	815
30.4	Treatable Exceptions in Native SQL	816

31 Data Clusters 817

31.1	Overview	817
31.2	Creating Data Clusters	817
31.2.1	Defining the Data Cluster	818
31.2.2	Defining the Storage Medium	819
31.2.3	Controlling Compression	824
31.2.4	Treatable Exceptions During the Export of Data Clusters	824
31.3	Reading Data Clusters	825
31.3.1	Specifying the Data Objects to Be Read	825
31.3.2	Defining the Storage Medium	827
31.3.3	Conversion Additions	831

31.3.4	Text Language Rule	837
31.3.5	Treatable Exceptions During the Import from Data Clusters	839
31.4	Reading the Table of Contents of a Data Cluster	839
31.5	Deleting a Data Cluster	841
31.6	Deleting a Data Cluster in the ABAP Memory	842

32 The ABAP File Interface 843

32.1	Overview	843
32.1.1	Addressing Files	843
32.1.2	Authorizations for Accessing Files	843
32.1.3	Locks	845
32.1.4	The File Interface in Unicode Programs	846
32.1.5	File Size	847
32.1.6	File Interface Statements	847
32.2	Opening a File	847
32.2.1	Definition of the Access Type	848
32.2.2	Defining the Storage Type	850
32.2.3	Specifying the Position	854
32.2.4	Operating System-Dependent Additions	855
32.2.5	Error Handling	857
32.2.6	Treatable Exceptions When Opening Files	859
32.3	Writing a File	860
32.3.1	The Influence of the Access Type	860
32.3.2	The Influence of the Storage Type	861
32.3.3	Restricting the Number of Transferred Characters or Bytes	862
32.3.4	Preventing the Appending of an End-of-Line Marking	863
32.3.5	Treatable Exceptions When Writing in Files	863
32.4	Reading a File	864
32.4.1	The Influence of the Access Type	864
32.4.2	The Influence of the Storage Type	865
32.4.3	Restricting the Number of Imported Characters or Bytes	866
32.4.4	Determining the Number of Imported Characters or Bytes	868
32.4.5	Treatable Exceptions When Reading Files	868
32.5	Determining the Attributes of an Opened File	869
32.5.1	Determining the Position of the File Pointer	869
32.5.2	Determining Other Attributes	870
32.5.3	Treatable Exceptions When Determining File Attributes	872
32.6	Changing the Attributes of an Opened File	872
32.6.1	Defining the Position of the File Pointer	873
32.6.2	Changing Further Attributes	874
32.6.3	Treatable Exceptions When Changing File Attributes	876
32.7	Changing the Size of a File	877
32.8	Closing a File	878
32.9	Deleting a File	879

33 Data Consistency 881

33.1	Overview	881
33.2	Database LUW	881
33.2.1	Database Commit	882
33.2.2	Database Rollback	883
33.3	SAP LUW	883
33.3.1	SAP Commit	884
33.3.2	SAP Rollback	886
33.3.3	Local Updating	887
33.4	Database Locks	888
33.5	SAP Locks	888
33.5.1	Imposing SAP Locks	889
33.5.2	Lifting SAP Locks	889
33.6	Authorization Check	890

Part 11 Program Parameters

34 Parameters in the SAP Memory 895

34.1	Overview	895
34.1.1	SPA/GPA parameters	895
34.1.2	Administrating SPA/GPA parameters	895
34.1.3	SPA/GPA parameters and dynpro fields	896
34.1.4	Statements for SPA/GPA Parameters	896
34.2	Setting Parameters	896
34.3	Reading Parameters	898

35 Language Environment 901

35.1	Overview	901
35.1.1	Text Pools	901
35.1.2	Text Environment	901
35.1.3	Country Identification for List Processing	903
35.1.4	Statements for the Language Environment	904
35.2	Setting the Text Pool of a Language	904
35.3	Setting the Text Environment	905
35.4	Determining the Text Environment	910
35.5	Setting Country Identification	911

36 Date and Time Information 913

36.1	Overview	913
36.1.1	System Fields for Date and Time	913
36.1.2	Time Stamps	914
36.1.3	Statements for Date and Time	917
36.2	Supplying System Fields for Date and Time	917
36.3	Creating Current Time Stamp	918
36.4	Converting Time Stamp Into Local Time	919
36.5	Converting Local Time Into a Time Stamp	921

Part 12 Program Processing

37 Testing and Checking Programs 927

37.1	Overview	927
37.2	Checkpoints	927
37.2.1	Defining Assertion	927
37.2.2	Setting Breakpoint	929
37.3	Runtime Measurement	931
37.3.1	Relative Program Runtime	931
37.3.2	Setting Time Resolution	933
37.4	Measuring Section for Runtime Analysis	934
37.5	Deactivating the Extended Program Check	935

38 Dynamic Program Development 937

38.1	Overview	937
38.2	Dynamic Subroutine Pool	937
38.2.1	Additions for Error Exceptions	939
38.3	Reading an ABAP Program	943
38.4	Checking Syntax	944
38.4.1	Determining the Attributes of the Syntax Check	945
38.4.2	Additions for Error Handling	946
38.5	Creating or Overwriting an ABAP Program	948
38.5.1	Determine Program Attributes	949
38.6	Reading a Text Pool	953
38.7	Creating or Overwriting a Text Pool	954
38.8	Calling the ABAP Editor	956

Part 13 External Programming Interfaces

39 Remote Function Call 961

39.1	Overview	961
39.1.1	RFC Interface	961
39.1.2	RFC Destination	962
39.1.3	RFC Context	963
39.1.4	RFC Exceptions	964
39.1.5	RFC System Fields	964
39.1.6	RFC—Authorization	965
39.1.7	RFC Restrictions	965
39.1.8	Statements of the RFC Interface	966
39.2	Calling Remote Functions	966
39.2.1	Synchronous Remote Function Call	967
39.2.2	Asynchronous Remote Function Call	968
39.2.3	Transactional Remote Function Call	973
39.2.4	Remote Function Call Example	975

40 ABAP and XML 979

40.1	XSL Transformations	979
40.1.1	XSL Transformations in the Repository	979
40.2	Canonical XML Representation	980
40.2.1	General asXML Format	980
40.2.2	asXML Format for Named Data Objects with the Exception of Reference Variables	981
40.2.3	asXML Format for Reference Variables and Referenced Objects	983
40.3	Simple Transformations	990
40.4	Calling an XSL or ST Transformation	990
40.4.1	Specifying the Transformation	992
40.4.2	Transformation Source	992
40.4.3	Result of a Transformation	993
40.4.4	Parameters for an XSL Transformation	995
40.4.5	Passing External Objects to an XSL Transformation	996
40.4.6	Controlling the Transformation	997
40.5	Example of an XSL Transformation	998
40.6	Simple Transformation Example	1001

41 OLE Interface 1005

41.1	Creating an Automation Object	1006
41.2	Calling an Automation Method	1007

41.3	Reading the Attributes of an Automation Object	1010
41.4	Setting the Attributes of an Automation Object	1011
41.5	Releasing an Automation Object	1012

Part 14 Obsolete Statements

42	Obsolete Statements	1017
-----------	----------------------------	-------------

42.1	Overview	1017
42.2	Obsolete Syntax	1017
42.3	Obsolete Modularization	1018
42.3.1	Obsolete Event Block	1018
42.4	Obsolete Declarations	1018
42.4.1	Obsolete Interface Work Areas	1018
42.4.2	Obsolete Declarations of Internal Standard Tables	1021
42.4.3	Obsolete Declarations of Special Internal Tables	1023
42.4.4	Obsolete Note for the Extended Program Check	1025
42.5	Obsolete Object Creation	1026
42.6	Obsolete Program Call	1028
42.7	Obsolete Exiting of a Program	1030
42.8	Obsolete Program Flow Control	1031
42.8.1	Obsolete Relational Operators	1031
42.8.2	Obsolete Branching	1031
42.9	Obsolete Assignments	1033
42.9.1	Obsolete Assignment of a Percentage Subfield	1033
42.9.2	Obsolete Conversion	1034
42.9.3	Obsolete Temporary Storage of Data Objects	1035
42.10	Obsolete Calculation Statements	1036
42.10.1	Addition of Field Sequences in the Memory	1036
42.10.2	Adding Component by Component	1038
42.10.3	Subtracting Component by Component	1039
42.10.4	Multiplying Component by Component	1040
42.10.5	Dividing Component by Component	1041
42.10.6	Obsolete Calculations During List Creation	1041
42.11	Obsolete Character-String Processing	1044
42.11.1	Obsolete Translation	1044
42.11.2	Obsolete Replacement	1046
42.11.3	Complement on Nine of a Date	1048
42.12	Obsolete Processing of Internal Tables	1049
42.12.1	Obsolete Key Specification When Reading Rows	1049
42.12.2	Obsolete Assignment to Table Lines	1052
42.12.3	Obsolete Form of the PROVIDE Statement	1054
42.13	Contexts	1056
42.13.1	Overview	1056

42.13.2	Creating Instances of Contexts	1057
42.13.3	Providing Contexts with Key Values	1058
42.13.4	Querying Contexts	1058
42.14	Obsolete Statements in the Flow Logic of Dynpros	1060
42.14.1	Checking Values in the Flow Logic	1060
42.14.2	Processing Step Loops	1063
42.15	Obsolete Statements in List Processing	1068
42.15.1	Obsolete Formatting Statements	1068
42.15.2	Obsolete Print Parameters	1070
42.15.3	Obsolete Creation of a Spool Task	1071
42.16	Obsolete Database Accesses	1072
42.16.1	Obsolete Reading of a Row	1072
42.16.2	Obsolete Sequential Reading of Several Rows	1074
42.16.3	Obsolete Reading of Several Rows into an Internal Table	1075
42.16.4	Obsolete Short Forms in Open SQL	1077
42.17	Obsolete External Programming Interface CPC-I	1078
42.17.1	Connection Steps	1079
42.17.2	Other Additions	1082
42.18	Obsolete Call of a Text Editor	1086

Part 15 Appendix

A Conversion Rules for Assignments 1091

A.1	Overview	1091
A.2	Conversion Rules for Elementary Data Types	1091
A.2.1	Presenting Numeric Values in Character-Like Fields	1092
A.2.2	Conversion Table for Source-Field Type c	1093
A.2.3	Conversion Table for Source-Field Type d	1094
A.2.4	Conversion Table for Source-Field Type f	1096
A.2.5	Conversion Table for Source Field of Types i, b, or s	1097
A.2.6	Conversion Table for Source-Field Type n	1098
A.2.7	Conversion Table for Source-Field Type p	1099
A.2.8	Conversion Table for Source-Field Type string	1100
A.2.9	Conversion Table for Source-Field Type t	1101
A.2.10	Conversion Table for Source-Field Type x	1102
A.2.11	Conversion Table for Source-Field Type xstring	1103
A.3	Conversion Rules for Structures	1104
A.3.1	Conversion Between Flat Structures	1105
A.3.2	Conversion Between Flat Structures and Single Fields	1108
A.4	Conversion Rules for Internal Tables	1109
A.5	Assignments Between Reference Variables	1110
A.5.1	Static and Dynamic Type	1110
A.5.2	Up Cast and Down Cast	1111
A.5.3	Assignments Between Data Reference Variables	1112
A.5.4	Assignments Between Object Reference Variables	1113

B Language-like Classes and Interfaces 1115

B.1	Auxiliary Classes	1115
B.1.1	Run Time Type Services (RTTS) Classes	1115
B.1.2	Classes for Converting External Data Formats	1116
B.1.3	Class for Extreme Values of Data Objects	1117
B.1.4	Class for the Properties of Characters	1117
B.1.5	Classes for Mathematical Operations	1117
B.1.6	Class for Time Stamps	1118
B.1.7	Classes for Data Clusters	1118
B.1.8	Class for Transactions	1118
B.1.9	Class for Formatting Lists	1119
B.1.10	Classes for Compressing Data	1119
B.1.11	Class for Runtime Measurements	1119
B.1.12	Class for Weak References	1120
B.2	Interface for the Serialization of Objects	1120
B.3	Shared Objects	1120
B.4	Object Services	1121
B.5	JavaScript Integration	1121

C Language-like Function Modules 1123

C.1	Function Modules for Print Parameters	1123
C.1.1	GET_PRINT_PARAMETERS	1123
C.1.2	SET_PRINT_PARAMETERS	1126
C.2	Function Modules for Files on the Presentation Server	1126
C.3	Function Modules for Calling Logical Databases	1127

D Predefined Treatable Exceptions 1129

D.1	Overview	1129
D.2	Predefined Exception Classes	1129
D.3	Catchable Runtime Errors	1131
D.3.1	Exception Group for Arithmetic Errors	1132
D.3.2	Exception Group for Conversion Errors	1133
D.3.3	Exception Group for Errors When Creating Data Objects	1134
D.3.4	Exception Group for Errors When Creating Instances of Classes	1134
D.3.5	Exception Group for Errors When Accessing Data Objects	1134
D.3.6	Exception Group for Errors in Dynamic Method Calls	1135
D.3.7	Exception Group for Errors When Accessing Files	1136
D.3.8	Exception Group for Errors When Accessing Data Clusters	1136
D.3.9	Exception Group for Errors in the Language Environment	1137

D.3.10 Exception Group for Errors with Remote Function Calls 1137

D.3.11 Catchable Runtime Errors That Are Not Assigned
to Any Group 1137

E	Glossary	1139
----------	-----------------	-------------

F	Notes on the CD-ROMs	1195
----------	-----------------------------	-------------

	Index	1197
--	--------------	-------------

40 ABAP and XML

With the `CALL TRANSFORMATION` statement (see Section 40.4), you can convert ABAP data into the XML format and vice versa. Transformation programs of the following types are called: XSL transformations or Simple Transformations (as of Release 6.40).



40.1 XSL Transformations

An XSL transformation is a program in the repository that is written in XSLT (Extensible Stylesheet Language Transformation program) and used for the transformation of XML documents. When calling an XSL transformation using the `CALL TRANSFORMATION` statement, you can also directly convert ABAP data into XML and vice versa. For this purpose, a serialization or de-serialization is carried out implicitly.

In the case of transformations that use ABAP data as a source, the ABAP data is first serialized into a canonical XML representation (asXML, see 40.2), which then serves as the actual source for the XSL transformation. In the case of transformations that expect ABAP data as a result, the result of the XSL transformation is de-serialized into the ABAP data. As a prerequisite for de-serialization is the result must take the form of a canonical XML representation.

As of Release 6.10, the ABAP runtime environment contains an XSLT processor for executing the transformations. It supports almost all XSLT statements and provides enhancements (so-called extension instructions) such as the possibility to call ABAP methods from XSLT programs.

40.1.1 XSL Transformations in the Repository

XSL transformations that can be called with a `CALL TRANSFORMATION` statement must exist in the repository as XSLT programs. To create and edit XSLT programs in the Object Navigator in the ABAP Workbench, choose **Edit Object • More ... • Transformation** (or **XSLT program** before Release 6.40 and **XSL transformation** before Release 6.20) and choose XSLT program.

SAP delivers the identity transformation under the name ID. If you perform an identity transformation from XML to XML, the result is a copy of the source document. If you perform an identity transformation from ABAP to XML, this results in a canonical XML representation (asXML) of the ABAP data (explicit serialization). An identity transformation from

XML to ABAP transforms a canonical XML representation to ABAP data (explicit de-serialization).

40.2 Canonical XML Representation

The canonical XML representation is the format of an XML document that results from a serialization of ABAP data or that is required for a de-serialization. This format is also referred to as asXML (ABAP Serialization XML). The canonical XML representation supports all ABAP data types.

The asXML format is significant in the following cases:

- If you have written any XSL transformations of ABAP data into an XML format, the asXML format of the serialization result must be known.
- If you want to create external XML documents that can be de-serialized into ABAP data, they must be in an asXML format.

Note

The asXML format of serialized ABAP data or objects can be created and examined using the predefined identity transformation ID.

40.2.1 General asXML Format

The following lines show the general format of the canonical XML representation¹ without the XML header; line breaks and indents are included for clarification purposes only. A detailed example can be found in Section 40.5.

```
<asx:abap version = "1.0"
      xmlns:asx = "http://www.sap.com/abapxml">
  <asx:values>
    <bn1>...</bn1>
    ...
    <bnn>...</bnn>
  </asx:values>
  <asx:heap>
    ...
  </asx:heap>
</asx:abap>
```

The root element of an asXML documents is `abap` in the namespace (XML Namespace) `http://www.sap.com/abapxml`. The optional attribute `ver-`

¹ The asXML format is a general format that cannot be completely defined with an XML pattern. The reason for this is that various ABAP types are referred to.

sion currently always has the value "1.0" and is intended for future enhancements of asXML. The root element `abap` must contain the sub-element `values` of the same namespace. The sub-elements `bn1` of `values` represent the ABAP data objects that are specified as `e1`, `e2`, ... in the `source` addition to the `CALL TRANSFORMATION` statement or as `f1`, `f2`, ... in the `result` addition (see Section 40.4). The names of the elements `bn1`, `bn2`, ... are the names specified there in uppercase. The text contents of the elements `<bn1>...</bn1>` (or `<bn1 ... />`), ... represent the contents of all named data objects with the exception of reference variables. The latter are represented by elements without text contents but with a special attribute (see Section 40.2.3.1). The optional element `heap` contains the contents of referenced anonymous data objects and objects (see Sections 40.2.3.2 and 40.2.3.3).

With the exception of the special cases in Table 40.1, the names of the element `bn1`, `bn2`, ... contain only capital letters. The names `bn1`, `bn2`, ... (or components of structures or objects, see Sections 40.2.2.2 and 40.2.2.3) specified in the `source` and `result` additions to the `CALL TRANSFORMATION` statement can only be used as (uppercase) names for XML elements if they consist solely of the characters "a" to "z", "A" to "Z", "0" to "9", or "_"; the first character must be a letter or "_". Other characters are replaced according to Table 40.1.

Character in the ABAP name	Replacement character in the XML name
ASCII character other than "a" to "z", "A" to "Z", "0" to "9", or "_" and character "0" to "9" as first character.	"_--hex(c)," where hex(c) is the two-digit hexadecimal representation of the ASCII code of the character c.
"/"	"_-"
"xml" as the first three characters in any combination of uppercase and lowercase	"x-ml" in a corresponding combination of uppercase and lowercase

Table 40.1 Replacement Rules for Charaters in ABAP Names

40.2.2 asXML Format for Named Data Objects with the Exception of Reference Variables

Named data objects, except for reference variables, are represented as the text contents of the elements `<bn1>...</bn1>`, ... The representation of named data objects in `<bn1>...</bn1>`, ... depends on the relevant ABAP data type.

40.2.2.1 Elementary Data Types

The asXML representation of elementary data objects with predefined ABAP types from Table 5.2 corresponds to the canonical representation of XML pattern data types (<http://www.w3.org/TR/xmlschema-2/#built-in-datatypes>, see Table 40.2), where date and time are represented according to ISO-8601, and binary data is represented using Base 64 encoding.

ABAP type	ABAP example	XML pattern type	XML example
c	" Hi"	string	" Hi"
d	"20020204"	date	"2002-02-04"
f	-3.140...0E+02	double	"-3.14E2"
i, b, s	-123	int, unsignedByte, short	"-123"
n	"001234"	string (pattern [0-9]+)	"001234"
p	-1.23	decimal	"-1.23"
string	" Hello "	string	" Hello "
t	"201501"	time	"20:15:01"
x	"ABCDEF"	base64Binary	"q83v"
xstring	"456789AB"	base64Binary	"RweJqw=="

Table 40.2 Canonical XML Representation (asXML) of Predefined ABAP Types

40.2.2.2 Structures

In asXML, the components of an ABAP structure are represented as a sequence of sub-elements of the structure element. The content of each sub-element corresponds to the canonical representation of the component value. The name of each sub-element is the name of the corresponding component. In the case of serialization, the sub-elements are represented in the order of the components in the structure. When the asXML representation of a structure is de-serialized, the order of the sub-elements is irrelevant and excess XML elements are ignored. Components of the structure for which there is no sub-element remain initial.

40.2.2.3 Internal Tables

In asXML, the rows of an internal table are represented as a sequence of sub-elements of the table element. The content of each sub-element corresponds to the canonical representation of the row value. The name of a

sub-element is irrelevant. If the canonical XML representation is created by serialization and the row type refers to the ABAP Dictionary, the name there is used; otherwise, the name `item` is used. Any table kind is allowed. During serialization, no information about the table kind is transferred to the XML document. If the target field of an XSL transformation is a sorted table, the rows are sorted accordingly during de-serialization.

40.2.3 asXML Format for Reference Variables and Referenced Objects

Anonymous data objects and instances of classes (objects) are addressed in ABAP exclusively by means of references in reference variables. The corresponding asXML format is made up of sub-elements of `values` for named reference variables (see Section 40.2.3.1) and of sub-elements of `heap` (see Sections 40.2.3.2 and 40.2.3.3) for the referenced objects. The link between the reference elements and the object elements is set up by means of an XML reference mechanism, whereby a referenced object in the same XML document is identified with a key. The dynamic type of the reference variables for the object elements under `heap` is specified when serialization takes place, so that de-serialization is unambiguous.

40.2.3.1 Named Reference Variables

A named reference variable is the only attribute of the corresponding sub-element of `values` that is displayed without textual content. An attribute of a reference variable has the name `href` and the content "#key", where `key` is the unique key of an object in the element `heap`. An element of an initial reference does not have a `href` attribute or any other content. During serialization, the ABAP runtime environment sets the key `key`; any key is possible for de-serialization.

40.2.3.2 Anonymous Data Objects

An anonymous data object that is a sub-element of `heap` is displayed as follows:

```
<asx:heap xmlns:nspace ...>
  <type id = "key" attr="...">...</type>
</asx:heap>
```

The value of a sub-element of this kind is displayed in the asXML display for named data objects (see Table 40.2) or for named reference variables (see Section 40.2.3.1). If the anonymous data object itself is a non-initial

reference variable, it references a further element of `heap` according to the rules above. The element name `type` is the data type of the data object (or the dynamic type of the reference variables) specified as the XML schema type name from the name range `nspc` (see Table 40.3). Attributes `attr` may define technical characteristics of the type. The mandatory attribute `attri` contains the unique key `key` of the element, which is used to reference it from the display of the corresponding reference variables in `values` or `heap`.

The XML schema type name is constructed according to the following hierarchy:

1. If the data type of the data object is defined in the ABAP Dictionary, the XML schema type name is the name of the data type from the ABAP Dictionary in the corresponding name range (see Table 40.3).
2. If the data type is an elementary ABAP type, the XML schema type name is specified in Table 40.4.
3. If the data type is defined as a component of a global or local class or interface, the XML schema type name comprises the name of the class or interface and the name of the data type separated by a period (.). The corresponding name range (see Table 40.3) indicates whether the data type is a component of a global or local class or of an interface.
4. If the data type is a generic reference type defined with `REF TO data` or `REF TO object`, the XML schema type name is `refData` or `refObject`. Both of these have the name range `http://www.sap.com/abapxml/types/built-in`.
5. Otherwise, the XML schema type name is the name of a data type defined with `TYPES` and the corresponding name range (see Table 40.3) indicates where the data type is defined.

Before an XML schema type name can be constructed, the data type of the data object must have a name that can be used statically. If the data type only exists as a property of a data object and therefore only has a technical name (compare Section 6.1.4), a treatable exception takes place during serialization.

Table 40.3 indicates the name ranges for the XML schema type names; types in the first column stand for `http://www.sap.com/abapxml/types`. The name ranges indicate where a data type is defined. Characters other than "a" to "z," "A" to "Z," "0" to "9," "_" or "-" are displayed as "!hex(c)" in the names `prg`, `cpool`, `fpool`, `tpool`, `meth`, `func`, `form` and

class, where hex(c) is the two-character hexadecimal display of the ASCII code for the character "c".

Name range	Location of definition
types/dictionary	ABAP Dictionary
types/program/prg	ABAP program prg
types/class-pool/cpool	Class pool cpool
types/type-pool/tpool	Type group tpool
types/function-pool/fpool	Function group fpool
types/function/func	Function module func
types/program.form/prg/frm	Subroutine frm in program prg
types/function-pool.form/fpool/frm	Subroutine frm in function group fpool
types/method/class/meth	Method meth of a global class class
types/program.method/prg/class/meth	Method meth of a local class class in program prg
types/class-pool.method/cpool/class/meth	Method meth of a local class class in a class pool cpool
types/function-pool.method/fpool/class/meth	Method meth of a local class class in function group fpool

Table 40.3 Name ranges for XML schema type names, whereby types in the first column stands for `http://www.sap.com/abapxml/types`

The following table lists the XML schema type names for elementary ABAP types. These are slightly different from the canonical XML schema data types from Table 40.2, since the data type of anonymous data objects must be specified in full. The name ranges `namespace` for the elementary ABAP types for anonymous data objects are either `xsd="http://www.w3.org/2001/XMLSchema"` for general schema types or `abap="http://www.sap.com/abapxml/types/built-in"` for special ABAP schema types for which some technical attributes must be specified.

ABAP type	XML schema type name	Attributes
c	abap:string	maxLength
d	abap:date	—
f	xsd:double	—

Table 40.4 XML Schema Type Names for Elementary ABAP Types

ABAP type	XML schema type name	Attributes
i, b, s	xsd:int, xsd:unsigned Byte, xsd:short	–
n	abap:digits	maxLength
p	abap:decimal	totalDigits, fractionDigits
string	xsd:string	–
t	abap:time	–
x	abap:base64Binary	maxLength
xstring	xsd:base64Binary	–

Table 40.4 XML Schema Type Names for Elementary ABAP Types (cont.)

The attribute `maxLength` defines the length for ABAP types with a generic length. The XML schema type `abap:digits` restricts the value range for an element to digits. The XML schema type `abap:decimal` specifies the length and fractional portions via the attributes `totalDigits` and `fractionDigits`. The length specification `totalDigits` defines the number of places between 1 and 31. In ABAP programs, the length of data objects of the type `p` is specified in bytes and the number of decimal places is calculated from $2 \times \text{len} - 1$ (see Table 5.2). This means that the value of `totalDigits` is always odd in serialization. During de-serialization, an even value of `totalDigits` is implicitly increased by one.

Instances of Classes

The instance of a class (object) as a sub-element of `heap` is displayed as follows:

```
<asx:heap xmlns:namespace ...>
  <class id = "key">
    <part classVersion = "...">
      <name>...</name>
    </part>
    ...
  </class>
</asx:heap>
```

The element name `class` is the XML schema type name of the class for the object (or the dynamic type of the reference variables) from the name range `namespace` (see Table 40.5) in block capitals. The mandatory attribute `id` contains the unique key `key` of the element, which is used to reference it when the corresponding reference variables are displayed in `values`.

The sub-elements `<part>...</part>` contain the values of the instance attributes for individual object parts as sub-elements `<name>...</name>`. The individual object parts are defined by the classes in the current inheritance hierarchy that can be serialized (see below).

The name range for the class name indicates where the class is defined. Table 40.5 lists the possible name ranges; `classes` in the first column stands for `http://www.sap.com/apapxml/classes`. The substitution rule for the name ranges in Table 40.3 also applies to the names `prg`, `cpool` and `fpool`.

Name range	Location of definition
<code>classes/global</code>	Class library
<code>classes/program/prg</code>	Program <code>prg</code>
<code>classes/class-pool/cpool</code>	Class pool <code>cpool</code>
<code>classes/function-pool/fpool</code>	Function group <code>fpool</code>

Table 40.5 Name ranges for class names, whereby `classes` in the first column stands for `http://www.sap.com/apapxml/classes`

The values of the values of a class instance that can be serialized (instance attributes or output parameters for a special method, see below) are displayed as the content or as an attribute of `<name>...</name>` in the asXML display for named data objects (see Table 40.2) or for reference variables, wherein `name` is the name of an instance attribute or an output parameter in block capitals. If the object is an interface attribute, the name is preceded by the name of the interface separated by a period (.) in order to distinguish it from another class attribute of the same name. The substitution rules from Table 40.1 apply for the names.

The values of a class instance that can be serialized are defined by implementing the system interface `IF_SERIALIZABLE_OBJECT` in the class (see Section B.2). If the class does not implement the interface `IF_SERIALIZABLE_OBJECT`, the element `class` does not contain any sub-elements. All the instance attributes of a class in which the interface `IF_SERIALIZABLE_OBJECT` is implemented are serialized and de-serialized to the interface by default. You can change this behavior by declaring special utility methods (see below). Static attributes are neither taken into account during serialization nor during de-serialization (with the exception of the special constant `SERIALIZABLE_CLASS_VERSION`, see below).

Standard Behavior

If the class implements the interface `IF_SERIALIZABLE_OBJECT`, the element `<class>...</class>` contains at least one subelement `<part>...</part>`. These sub-elements correspond to individual object parts that can be serialized and contain the presentations of the instance attributes for the corresponding object part in an asXML format. An object part is defined by the class in which instance attributes are declared or in which an interface containing instance attributes is integrated. A object class that can be serialized contains an object part for itself as well as object parts for all superclasses in the current path in the inheritance tree, up to and including the class that implements the interface `IF_SERIALIZABLE_OBJECT`. The name `part` is the name of the class in question. If it is a local class, its name is preceded by the prefix `local` separated by a period (.) to distinguish it from a global class of the same name. Object parts of superclasses in which the interface `IF_SERIALIZABLE_OBJECT` is not implemented cannot be serialized and do not have a corresponding sub-object `part`. This means that a class in which the interface `IF_SERIALIZABLE_OBJECT` is not implemented (neither in the class itself nor in a superclass) creates a blank XML element `class` during serialization.

During serialization, the XML elements `part` of the object parts are created from the superclasses to the subclasses and the XML elements of the instance attributes are created as standard in the order in which they are declared in the class.

De-serialization creates an object in the corresponding class but the instance constructor is not executed. All instance attributes have their initial values or the start values specified with the `VALUE` addition for the `DATA` statement after the object creation. The values of the corresponding XML elements are entered in the instance attributes by default; the order of the object parts and attributes is irrelevant. Instance attributes without a corresponding XML element retain their value. Excess XML elements are ignored if they do not belong to a name range; otherwise, they create a treatable exception. When an element without `part` sub-elements is de-serialized, the system does not create an object but initializes the target reference variable.

If a class implements the interface `IF_SERIALIZABLE_OBJECT`, you can declare the private constant `SERIALIZABLE_CLASS_VERSION` of the type `i` in each object part; that is, each class involved in the inheritance tree. During serialization, the value of the constant is assigned to the attribute

`classVersion` of the XML element `part`. A treatable exception is created during de-serialization by default if the value of the attribute does not match the value of the constant in the class specified. An object can only be de-serialized if the values match or if there is neither an attribute nor a constant. You can change this system behavior by declaring special utility methods.

Modified Behavior

By default, all the instance attributes for an object part are serialized as standard regardless of their visibility, and the version of the class is checked. To change this behavior, you can declare and implement the instance methods `SERIALIZE_HELPER` and `DESERIALIZE_HELPER` in the relevant class for each object part. These methods can only be declared as private instance methods in classes that implement the interface `IF_SERIALIZABLE_OBJECT`. If you declare one of the methods, you must also declare the other and the interface must be defined as follows for the syntax check:

- ▶ The method `SERIALIZE_HELPER` can only have output parameters, and the method `DESERIALIZE_HELPER` can only have input parameters with non-generic typing.
- ▶ There must be an input parameter of the method `DESERIALIZE_HELPER` with the same name for each output parameter of the method `SERIALIZE_HELPER` with the same typing. Additional input parameters for the method `DESERIALIZE_HELPER` must be optional.
- ▶ The method `SERIALIZE_HELPER` must not have an output parameter with the name `SERIALIZABLE_CLASS_VERSION`, and the method `DESERIALIZE_HELPER` can have an optional input parameter of this name that is of type `i`. This parameter is supplied with the value of the attribute `classVersion` of the element `part` during de-serialization, and the standard check on the version (see above) is skipped.

If the methods `SERIALIZE_HELPER` and `DESERIALIZE_HELPER` are declared in an object part, the instance attributes of the object part are not serialized and de-serialized. Instead, the method `SERIALIZE_HELPER` is executed during serialization and the values of all the output parameters are written in the asXML format as sub-elements to the corresponding element `part` in the specified order. Here, the name of a sub-element is the name of the corresponding output parameter in block capitals. The method `DESERIALIZE_HELPER` is called during de-serialization and the values of the sub-elements for the corresponding element `part` are trans-

ferred to the input parameters of the method with the same names. The order in which they appear is irrelevant and excess XML elements are ignored.

40.3 Simple Transformations



Simple Transformations (ST) is an SAP programming language for describing transformations between ABAP data and XML formats. ST is restricted to the two modes of serialization (ABAP to XML) and de-serialization (XML to ABAP) of ABAP data, which are most important for data integration. Like in the more general XSLT, transformations from ABAP to ABAP and XML to XML are not possible in ST.

In comparison with XSLT, the main advantages of ST programs are as follows:

- ▶ ST programs are declarative and thus easier to read.
- ▶ ST programs only have serial access to the XML data and are therefore very efficient even with large data volumes.
- ▶ ST programs describe serialization and de-serialization simultaneously; that is, ABAP data serialized in XML with ST can also be de-serialized with the same ST program.

Simple Transformations that can be called using `CALL TRANSFORMATION` must be in the repository. In the Object Navigator of the ABAP Workbench, you can create and edit ST programs by choosing **Edit Object • More • Transformation** followed by **Simple Transformation**.

A detailed description of the language ST goes beyond the scope of this ABAP reference. For more information, please turn to the corresponding online help. You can find an introductory example in Section 40.6.

40.4 Calling an XSL or ST Transformation



CALL TRANSFORMATION

Calling an XSLT or ST program.

Syntax

```
CALL TRANSFORMATION transformation
                     [PARAMETERS parameters]
                     [OBJECTS objects]
```



```
[OPTIONS options]  
SOURCE source  
RESULT result.
```

This statement calls the specified XSL transformation (XSLT) or a simple transformation (ST, as of Release 6.40). The source of the transformation is specified after **SOURCE**, and the result is stored as specified after **RESULT**. Use **PARAMETERS** and **OBJECTS** to pass parameters to the transformation. Possible transformation types are:

- ▶ From XML to XML (only for XSLT)
- ▶ From XML to ABAP (for XSLT and ST)
- ▶ From ABAP to XML (for XSLT and ST)
- ▶ From ABAP to ABAP (only for XSLT)

The last two types are available only as of Release 6.20.

Treatable Exceptions

The common superclass of all exception classes for **CALL TRANSFORMATION** is **CX_TRANSFORMATION_ERROR** (as of Release 6.40). The corresponding runtime errors cannot be caught.



Exceptions with XSL-Transformations

All exception classes for XSL transformations are subclasses of **CX_XSLT_EXCEPTION**.



If an error occurs when passing an XML document or if another error is reported by the XSLT processor, an exception defined by the class **CX_XSLT_RUNTIME_ERROR** is triggered. If the calling of an ABAP method from the XSLT program leads to an error, an exception defined by the class **CX_XSLT_CALL_ERROR** is triggered, whereby the attribute **PREVIOUS** points to the exception object of the original error.

If an XML document does not have the asXML format during the de-serialization, an exception defined by the class **CX_XSLT_FORMAT_ERROR** is triggered, where the attribute **TREE_POSITION** contains the error position. If, during serialization or de-serialization, invalid values or data types occur, exceptions defined by the classes **CX_XSLT_SERIALIZATION_ERROR** or **CX_XSLT_DESERIALIZATION_ERROR** are triggered, where the attribute **PREVIOUS** (if required) points to the exception object of the original error. The attribute **TREE_POSITION** contains the error position during de-serialization.

Exceptions with Simple Transformations

All exception classes for simple transformations are subclasses of CX_ST_ERROR.

40.4.1 Specifying the Transformation

Syntax of *transformation*

```
... trans | (name) ...
```

The name of the transformation can be specified either directly as `trans` or as a content of a character-type data object `name` in brackets. The specified transformation must exist as a XSLT program or as a simple transformation in the repository.

40.4.2 Transformation Source

Syntax of *source*

```
... { XML sxml }  
    | {{bn1 = e1 bn2 = e2 ...}|(stab)) ...
```

40.4.2.1 Transformation of an XML Document

When you specify `XML sxml`, the XML document contained in `sxml` is transformed in such a way that `sxml` can have one of the following forms:

- ▶ Data object of type `string` and `xstring` or as a standard table with flat character-type row type
- ▶ Interface reference variable of type `IF_IXML_ISTREAM` which points to an iXML input stream (only for XSLT)
- ▶ Interface reference variable of type `IF_IXML_NODE` which points to an iXML nodeset (only for XSLT)
- ▶ Class reference variable of type `CL_FX_READER`, which points to an XML reader (only for ST)

Note

The interfaces `IF_IXML_ISTREAM` and `IF_IXML_NODE` are components of the "Stream" and "DOM" packages of the iXML Library delivered by SAP.

40.4.2.2 Transformation of ABAP Data

Use `bn1 = e1`, `bn2 = e2`, ... or `(stab)` to specify the ABAP data `e1`, `e2`, ... to be transformed.



- ▶ When calling an XSLT program, the ABAP data are serialized into the canonical XML representation, which is then used as source of the XSL transformation. Use `bn1`, `bn2`, ... to specify the names of the XML elements meant to represent the ABAP data objects in the canonical XML presentation.
- ▶ When calling a simple transformation, the names `bn1`, `bn2`, ... are used in the transformation to access the ABAP data in a written way.

Instead of using a static parameter list, you also can pass the data objects dynamically as value pairs in the columns of an internal table `stab` which has the type `ABAP_TRANS_SRCBIND_TAB` from the ABAP type group.

The following data objects cannot be serialized and trigger a treatable exception:

- ▶ Data objects of type `n`, whose current content does not exclusively consist of numbers.
- ▶ Data objects of type `p`, whose current content does not represent a valid packed number.
- ▶ Data objects of type `d` and `t`, whose current content contains leading or trailing blanks and at the same time uses the separators ("-" or ":") according to ISO-8601 for the presentation.
- ▶ Data reference variables pointing to data objects, whose data type has only a technical name (see Section 40.2.3).

Data reference variables pointing to data objects that were not created with `CREATE DATA` are treated as initial reference variables during the serialization.

40.4.3 Result of a Transformation

Syntax of *result*

```
... { XML rxml }  
| {{bn1 = f1 bn2 = f2 ...}|(rtab)} ...
```

40.4.3.1 Transformation into an XML Document

When you specify XML `rxml`, a transformation into an XML is executed; the document is then placed into `rxml`, where `rxml` can be one of the following:

- ▶ A data object of type `string` and `xstring` or a standard table with a flat, character-type row type.
- ▶ An interface reference variable of type `IF_IXML_OSTREAM` which points to an IXML output stream (only for XSLT).
- ▶ An interface reference variable of type `IF_IXML_DOCUMENT` which points to an IXML document (only for XSLT),
- ▶ A class reference variable of type `CL_FX_WRITER`, which points to an XML writer (only for ST).

Notes

- ▶ The interfaces `IF_IXML_OSTREAM` and `IF_IXML_DOCUMENT` are components of the "Stream" and "DOM" packages of the iXML Library delivered by SAP.
- ▶ If you use the data type `xstring` for `rxml`, then the result is stored in the UTF-8 character representation. This is helpful, if the resulting XML-document is to be stored in a file (see Chapter 32).

40.4.3.2 Transformation into ABAP Data

Use `bn1 = f1, bn2 = f2, ...` or `(rtab)` to specify the ABAP target fields `f1`, `f2`, ... into which you want the XML data to be transformed.

- ▶ When calling an XSLT program, the result of the XSL transformation into ABAP data objects is de-serialized, provided that it is a canonical XML representation. You should use `bn1`, `bn2`, ... to specify the names of the XML elements that represent the ABAP data objects in the canonical XML representation, and use `f1`, `f2`, ... to specify the ABAP data objects of the appropriate data type into which you want to de-serialize them.
- ▶ When calling a simple transformation, in the transformation, the names `bn1`, `bn2`, ... are used for write access to the ABAP data.

Instead of using a static parameter list, the data objects can also be passed dynamically as value pairs in the columns of the internal table `rtab`, which has the type `ABAP_TRANS_RESBIND_TAB` of the ABAP type group.

An XML element must be convertible into the respective ABAP data objects, where instead of the usual conversion rules (see Appendix A) the following restrictions apply:

- ▶ De-serialization into too short data objects of data types `c` or `n` must never lead to a loss of data, except when for data type `c` only leading and trailing blanks are concerned and for data type `n` only leading zeros.
- ▶ Data must never be lost due to the de-serialization into a data object of data type `p` with too few decimal places.
- ▶ Data must never be lost due to the de-serialization into a too short data object of data type `x`.
- ▶ Structures cannot be converted into elementary data objects.

If an XML element cannot be converted into the ABAP data object, a treatable exception is triggered.

When de-serializing into a reference variable, this variable must be the same as or more general than the dynamic type of the object stored in the XML document. The allocated ABAP objects or instances of a class are created during the de-serialization.

40.4.4 Parameters for an XSL Transformation

Syntax of *parameters*

```
... {p1 = e1 p2 = e2 ...} |(ptab) ...
```

Use this addition to pass ABAP data objects `e1`, `e2`, ... as parameters `p1`, `p2`, ... to an XSL transformation. In Release 6.10, the data objects `e1`, `e2`, ... must be character-type, as of Release 6.20 all elementary data objects and object references are allowed.

Instead of using a static parameter list, you also can pass the parameters dynamically as value pairs in the columns of the internal table `ptab` which has the type `ABAP_TRANS_PARMBIND_TAB` from the ABAP type group.

The specified parameters must be defined in the XSL transformation as input parameters as follows:

```
<xsl:param name="..." type="..."/>
```

For the attribute `name`, enter the parameter name in uppercase. For the optional attribute `type`, specify one of the type indicators `string`, `num-`

ber, boolean, xstring, nodeset or object(...), where you must enter the name of a global ABAP class in the brackets after object.

If no type is specified in the XSL transformation, the data types of elementary parameters are mapped to XSL types according to Table 40.6.

ABAP Data Type	XSL Parameter Type
c, d, n, string	string
i, s, b, f, p	number
x, xstring	string, where the content is presented to the base of 64

Table 40.6 Mapping of ABAP Data Types on XSL Parameter Types

If during the XSL transformation the XSL types shown in Table 40.6 are specified explicitly, you must enter the matching elementary ABAP parameters which can be converted into the XSL type:

- ▶ The XSL type `boolean` expects ABAP parameters of the type `c` with the length 1. A space is interpreted as "false" and a different character is interpreted as "true."
- ▶ The XSL type `xstring` expects ABAP parameters of the type `x` or `xstring` and the display of the content is hexadecimal.
- ▶ The XSL types `nodeset` and `object` expect an object reference variable pointing to a class instance. The type `nodeset` expects appropriate object properties.

If a parameter does not match the XSL type, an untreatable exception is triggered. If a parameter defined in the XSL transformation is not passed, it is set to the default value in the transformation. A specified parameter that is not defined in the XSL transformation is ignored.

Note

The XSL types `string`, `number`, `boolean` and `nodeset` are XSL standard types, whereas `xstring` and `object` are special SAP extensions. The type `xstring` allows a hexadecimal display of byte chains instead of the presentation to the base of 64. The type `object` enables you to call ABAP methods from the XSL program.

40.4.5 Passing External Objects to an XSL Transformation

Syntax of *objects*

```
... {o1 = e1 o2 = e2 ... }|(otab) ...
```

You can use this addition to pass object references `e1`, `e2`, ... as external objects `o1`, `o2`, ... to an XSL transformation where you can call their methods.

Instead of using a static parameter list, you can also pass the objects dynamically as value pairs in the columns of the internal table `otab` which has the type `ABAP_TRANS_OBJBIND_TAB` from the ABAP type group.

Note

As of Release 6.20, the addition `OBJECTS` is obsolete and external objects are treated as parameters. Therefore, object references should be passed with the addition `PARAMETERS` (see Section 40.4.4).



40.4.6 Controlling the Transformation

Syntax of *options*

```
... a1 = e1 a2 = e2 ...
```



You can use this addition to specify the values `e1`, `e2`, ... for additional control options `a1`, `a2`, ... of the transformation. The values `e1`, `e2`, ... must be of the type `c` or `string`.

For `a1`, `a2`, ... you can specify the following values:

- `XML_HEADER` to control the output of the XML header in case of a transformation to XML and in case of storage in a data object of the type `c`, `string` or in an internal table.

Possible values	Meaning
no	No output of an XML header
without_encoding	Output of an XML header without specification of the encodings
full	Default setting, output of an XML header with specification of the encoding

- `DATA_REFS` to control the output of data references in case of a transformation from ABAP to XML.

Possible values	Meaning
no	Default for ST, no output of data references

Possible values	Meaning
heap	Default for XSLT and only possible there; output of referenced data as sub-elements of the asXML elements <code><asx:heap></code> .
embedded	Output of referenced data with the reference

- `INITIAL_COMPONENTS` to control the output of initial structure components in case of a transformation from ABAP to XML.

Possible values	Meaning
include	Default setting, output of initial components of structures
suppress	No output of initial components of structures

40.5 Example of an XSL Transformation

This example shows the serialization of data objects in a string `xmlstr` using the identical transformation ID. A date field `date`, a time field `time`, and a data reference variable `dref1` are serialized. The data reference variable points to an anonymous object reference variable, which in turn points to an object of the class `c2`. Objects serialized in this way can be stored persistently, for example in a data cluster. After the objects are imported from where they are stored, they are de-serialized into further data objects. Following de-serialization, `dref2` points to another anonymous reference variable, such as `dref1`. This anonymous data object and the instance of the class `c2` to which it points are generated during the de-serialization.

```

PROGRAM xmltst.

CLASS c1 DEFINITION.
  PUBLIC SECTION.
    INTERFACES if_serializable_object.
  PROTECTED SECTION.
    DATA carriers TYPE TABLE OF scarr.
ENDCLASS.

CLASS c2 DEFINITION INHERITING FROM c1.
  PUBLIC SECTION.
    METHODS constructor.
  PRIVATE SECTION.
    DATA lines TYPE i.
    METHODS: serialize_helper

```



```

        EXPORTING count TYPE i,
        deserialize_helper
        IMPORTING count TYPE i.
ENDCLASS.

CLASS c2 IMPLEMENTATION.
    METHOD constructor.
        super->constructor( ).
        SELECT * UP TO 2 ROWS
            FROM scarr
            INTO TABLE carriers.
    ENDMETHOD.
    METHOD serialize_helper.
        count = LINES( carriers ).
    ENDMETHOD.
    METHOD deserialize_helper.
        lines = count.
    ENDMETHOD.
ENDCLASS.

DATA: oref    TYPE REF TO object,
      dref1   LIKE REF TO oref,
      xmlstr  TYPE string,
      date    TYPE d,
      time    TYPE t,
      dref2   LIKE dref1.

...

CREATE DATA dref1 LIKE oref.
CREATE OBJECT dref1->* TYPE c2.

CALL TRANSFORMATION id
    SOURCE xml-dat = sy-datum
           xml-tim = sy-uzeit
           ref     = dref1
    RESULT XML xmlstr.

EXPORT obj = xmlstr TO DATABASE indx(hk)
    ID 'OBJECT'.

...

IMPORT obj = xmlstr FROM DATABASE indx(hk) ID 'OBJECT'.

CALL TRANSFORMATION id
    SOURCE XML xmlstr
    RESULT xml-dat = date

```

```

xmltim = time
ref     = dref2.

```

The XML document generated in the serialization has the content described below. In this description, line breaks and indents have been added. The element `values` contains the asXML representations of the three transferred data objects (see Section 40.2). In the names `X-MLDAT` and `X-MLTIM`, "xml" has been replaced according to Table 40.1. The attribute `href` of the element `REF` uses the key "d1" to refer to the representation of the corresponding anonymous data object in the element `heap`. This uses the key "o3" to refer to the representation of the instance of the class `c2`, which is also in the element `heap`. This representation is divided into the object parts for the classes `c1` and `c2`. The object part for `c1` contains the representation of the double-line structured internal table `carriers`. The object part for `c2` contains the representation for the output parameter `count` of the method `SERIALIZE_HELPER`.

```

<?xml version="1.0" encoding="iso-8859-1" ?>
<asx:abap xmlns:asx="http://www.sap.com/
abapxml" version="1.0">
  <asx:values>
    <X-MLDAT>2003-04-15</X-MLDAT>
    <X-MLTIM>14:57:53</X-MLTIM>
    <REF href="#d1" />
  </asx:values>
  <asx:heap
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:abap="http://www.sap.com/abapxml/types/built-in"
    xmlns:cls="http://www.sap.com/abapxml/classes/global"
    xmlns:dic="http://www.sap.com/abapxml/types/dictionary">
    <abap:refObject href="#o3" id="d1" />
    <prg:C2
      xmlns:prg="http://www.sap.com/abapxml/classes/program/
      XMLTST id="o3">
      <local:C1>
        <CARRIERS>
          <SCARR>
            <MANDT>000</MANDT>
            <CARRID>AA</CARRID>
            <CARRNAME>American Airlines</CARRNAME>
            <CURRCODE>USD</CURRCODE>
            <URL>http://www.aa.com</URL>
          </SCARR>
        </CARRIERS>
      </local:C1>
    </prg:C2>
  </asx:heap>
</asx:abap>

```

```

        <MANDT>000</MANDT>
        <CARRID>AB</CARRID>
        <CARRNAME>Air Berlin</CARRNAME>
        <CURRCODE>DEM</CURRCODE>
        <URL>http://www.airberlin.de</URL>
    </SCARR>
</CARRIERS>
</local.C1>
<local.C2>
    <COUNT>2</COUNT>
</local.C2>
</prg:C2>
</asx:heap>
</asx:abap>

```

40.6 Simple Transformation Example

Serialization of a nested structure. In the following ABAP program section, a nested structure `struc1` is serialized to `xml_string` with the Simple Transformation `ST_TRAFO` and de-serialized with the same transformation.

```

DATA: BEGIN OF struc1,
      col1(10) TYPE c VALUE 'ABCDEFGHJIJ',
      col2      TYPE i VALUE 111,
      BEGIN OF struc2,
        col1 TYPE d VALUE '20040126',
        col2 TYPE t VALUE '084000',
      END OF struc2,
    END OF struc1.

```

```

DATA: xml_string TYPE string,
      result LIKE struc1.

```

```

TRY.

```

```

    CALL TRANSFORMATION st_trafo
      SOURCE para = struc1
      RESULT XML xml_string.

```

```

    ...

```

```

    CALL TRANSFORMATION st_trafo
      SOURCE XML xml_string
      RESULT para = result.

```

```

CATCH cx_st_error.

```

...

ENDTRY.

The Simple Transformation ST_TRAFO has the following form:

```
<?sap.transform simple?>
<tt:transform template="temp"
  xmlns:tt="http://www.sap.com/transformation-templates"
  version="0.1">

  <tt:root name="PARAM"/>

  <tt:template name="temp">
    <X>
      <X1>
        <tt:value ref="PARAM.COL1" />
      </X1>
      <X2>
        <tt:value ref="PARAM.COL2" />
      </X2>
      <X3>
        <X1>
          <tt:value ref="PARAM.STRUC2.COL1" />
        </X1>
        <X2>
          <tt:value ref="PARAM.STRUC2.COL2" />
        </X2>
      </X3>
    </X>
  </tt:template>
</tt:transform>
```

The transformation consists of a template `temp` that defines the structure of the XML document and establishes relationships between value nodes and components of the structure. The result of the transformation is as follows (line breaks and indentations were inserted for clarification purposes):

```
<X>
  <X1>ABCDEFGH IJ</X1>
  <X2>111</X2>
  <X3>
    <X1>2004-01-26</X1>
    <X2>08:40:00</X2>
```

```
</X3>  
</X>
```

The conversion of the elementary data types is the same as for asXML (see Table 40.2). The reverse transformation generates the same content in the structure `result` as in `struc1`.

Index

#EC, pseudo comment 49
&, literal operator 117
' , text field literal 117
(,)
 arithmetic expression 395
 bit expression 400
 CALL METHOD 263
 logical expression 329
 offset/length specification 45
 SELECT INTO 762
*
 comment 49
 SELECT 749
 TABLES, obsolete 1020
 WRITE 653
**, WRITE 653
*-INPUT, FIELD 544
+, -, *, /, **, arithmetic operators 395
+, -, prefixes 394
,
 chained statement 48
 SELECT INTO 762
 WHERE IN 769
-, structure component selector 37
., statement 35
/
 namespace prefix 47
 SELECTION-SCREEN COMMENT 590
 SELECTION-SCREEN PUSHBUTTON 592
 SELECTION-SCREEN ULINE 589
 ULINE 673
 WRITE 653
:, chained statement 48
=
 CALL FUNCTION 276
 CALL METHOD 264
 COMPUTE 393
 conjunction operator 36
 EXPORT 818
 FORMAT 674
 IMPORT 825
 MOVE 362
 UPDATE 792
 WRITE 667

=, <>, <, >, <=, >=
 relational operators 36
 WHERE 767
=>, class component selector 39
->*, dereferencing operator 374
->, object component selector 38
><, =<, =>, relational operators, obsolete 1031
?=: MOVE 362
?TO, MOVE 362
[], table body 44
' , string literal 118
~, interface component selector 40

A

ABBREVIATED
 SEARCH 415
 SEARCH – internal table 482
abs, predefined function 109
ABSTRACT
 CLASS 158
 METHODS 175
ABSTRACT METHODS, INTERFACES 192
ACCEPT, COMMUNICATION 1080
ACCEPTING DUPLICATE KEYS, INSERT 789
ACCEPTING PADDING, IMPORT 832
ACCEPTING TRUNCATION, IMPORT 833
ACCORDING, ADD, obsolete 1036
ACCP, data type 98
acos, predefined function 110
ACTIVATION, SET HANDLER 293
ACTUAL LENGTH, READ DATASET 868
ADD
 statement 401
 statement, obsolete 1036
ADD-CORRESPONDING, statement, obsolete 1038
ADJACENT DUPLICATES, DELETE 475
AFTER INPUT, PROCESS 536
ALIASES, statement 195
ALL FIELDS
 DELETE 475
 READ TABLE 442
ALL INSTANCES, SET HANDLER 292

ALL METHODS ABSTRACT, INTERFACES 192

ALL METHODS FINAL, INTERFACES 192

ALL OCCURRENCES, REPLACE 418

ALL, subquery 775

ALLOCATE, COMMUNICATION 1080

AND

- Boolean operator 328
- WHERE 772

any table, generic data type 97

ANY TABLE, TYPES 128

any, generic data type 97

ANY, subquery 775

APPEND, statement 461

APPENDING

- FETCH 783
- OPEN DATASET 849
- SELECT INTO 763

ARCHIVE MODE, print parameter, obsolete 1071

ARCHIVE PARAMETERS

- NEW-PAGE 694
- SUBMIT 244

AREA HANDLE, CREATE OBJECT 224

AREA, GET CURSOR 570

AS

- INCLUDE STRUCTURE 145
- INCLUDE TYPE 145
- SELECT 749
- SELECT FROM 754

ASCENDING

- SELECT 780
- SORT – extract 492
- SORT – internal table 476
- SORT BY – extract 493
- SORT BY – internal table 478

asin, predefined function 110

ASSERT, statement 927

ASSIGN LOCAL COPY OF, statement, obsolete 1026

ASSIGN, statement 369

ASSIGNED, logical expression 321

ASSIGNING

- APPEND 465
- COLLECT 461
- INSERT 459
- LOOP AT 448
- MODIFY 469

READ TABLE 444

AT

- extract 495
- internal table 451
- TRUNCATE DATASET 877
- ULINE 673
- WRITE 653

AT LINE-SELECTION, statement 85

AT PF, statement, obsolete 1018

AT SELECTION-SCREEN, statement 77

AT USER-COMMAND, statement 86

atan, predefined function 110

ATTRIBUTES

- GET DATASET 870
- SET DATASET 874

AUTHORITY-CHECK, statement 890

AVG, SELECT 750

B

b, data type 93

BACK, statement 685

BACKGROUND TASK, CALL FUNCTION 973

BACKUP INTO, EDITOR-CALL, obsolete 1086

BACKWARD, SCROLL LIST 709

BEFORE OUTPUT, PROCESS 536

BEGIN OF

- CLASS-DATA 141
- CONSTANTS 142
- DATA 137
- DATA, obsolete 1022
- SELECTION-SCREEN 584
- STATICS 144
- TYPES 126

BETWEEN

- DESCRIBE DISTANCE 508
- logical expression 320
- PROVIDE 484
- PROVIDE, obsolete 1054
- WHERE 768

BIG ENDIAN, OPEN DATASET 852

BINARY MODE, OPEN DATASET 850

BINARY SEARCH, READ TABLE 439

BIT-AND, -NOT, -OR, -XOR, bit operators 399

BLOCK

- AT SELECTION-SCREEN 79

- SELECTION-SCREEN 596
- BLOCKS
 - SELECTION-SCREEN EXCLUDE 606
 - SELECTION-SCREEN INCLUDE 605
- BOUND, logical expression 321
- BOUNDS, PROVIDE 484
- BREAK-POINT, statement 929
- BYPASSING BUFFER, SELECT 760
- BYTE MODE, byte string processing 406
- BYTE-CA, -CN, -CO, -CS, -NA, -NS, relational operators 318
- C**
- c
 - data type 93
 - generic data type 97
- CA, comparison operator 316
- CALL CUSTOMER-FUNCTION, statement 282
- CALL DIALOG, statement, obsolete 1028
- CALL FUNCTION ... DESTINATION, statement 966
- CALL FUNCTION ... IN UPDATE TASK, statement 283
- CALL FUNCTION ... STARTING NEW TASK, statement 968
- CALL FUNCTION ... IN BACKGROUND TASK, statement 973
- CALL FUNCTION, statement 274
- CALL METHOD OF, OLE, statement 1007
- CALL METHOD, statement 262
- CALL SCREEN, statement 557
- CALL SELECTION-SCREEN, statement 639
- CALL SUBSCREEN, dynpro statement 554
- CALL TRANSACTION, statement 248
- CALL TRANSFORMATION, statement 990
- CALLING, CALL FUNCTION STARTING NEW TASK 970
- CASE
 - FIND 411
 - REPLACE 419
 - statement 333
 - TRANSLATE 430
- CASTING, ASSIGN 377
- CATCH SYSTEM-EXCEPTIONS, statement 354
- CATCH, statement 350
- ceil, predefined function 109
- CENTERED, WRITE 656
- CHAIN, dynpro statement 547
- CHAIN-INPUT
 - FIELD MODULE 543
 - MODULE 540
- CHAIN-REQUEST
 - FIELD MODULE 543
 - MODULE 540
- CHANGING
 - CALL FUNCTION 276
 - CALL FUNCTION DESTINATION 967
 - CALL METHOD 265
 - CLASS-METHODS 185
 - FORM 66
 - METHODS 170
 - PERFORM 288
 - RECEIVE RESULTS 972
- CHAR, data type 98
- CHARACTER MODE, character-string processing 406
- charlen, predefined function 111
- CHAR-TO-HEX MODE, IMPORT 836
- CHECK
 - loop 307
 - processing block 303
- CHECKBOX
 - PARAMETERS 616
 - WRITE 668
- CIRCULAR, SHIFT 423
- CLASS, statement 154
- class_constructor, static constructor 186
- CLASS-DATA, statement 141
- CLASS-EVENTS, statement 191
- CLASS-METHODS, statement 184
- CLASS-POOL, statement 59
- CLEANUP, statement 350
- CLEAR, statement 387
- CLIENT
 - DELETE 841
 - EXPORT 821
 - IMPORT 828
 - IMPORT DIRECTORY 839
- CLIENT SPECIFIED
 - DELETE 800
 - INSERT 787
 - MODIFY 797
 - SELECT 760
 - UPDATE 791

- clike, generic data type 97
- CLNT, data type 98
- CLOSE CURSOR, statement 784
- CLOSE DATASET, statement 878
- CLOSE, EXEC SQL 809
- CN, comparison operator 316
- cnt, AT 497
- CO, comparison operator 316
- CODE PAGE
 - OPEN DATASET 852
 - TRANSLATE, obsolete 1045
- CODEPAGE INTO, IMPORT 836
- COL_..., FORMAT 675
- COLLECT, statement 459
- COLOR
 - FORMAT 675
 - PRINT-CONTROL 700
 - WRITE 667
- COLUMN
 - SCROLL LIST 709
 - SET LEFT SCROLL-BOUNDARY 687
- COMMENT
 - SELECTION-SCREEN 590
 - SELECTION-SCREEN INCLUDE 603
- COMMIT WORK, statement 884
- COMMON PART, DATA 1018
- COMMUNICATION, statement, obsolete 1078
- COMPARING
 - DELETE 475
 - READ TABLE 442
- COMPONENT, ASSIGN 375
- COMPONENTS, DESCRIBE FIELD 500
- COMPRESSION, EXPORT 824
- COMPUTE, statement 393
- CONCATENATE, statement 408
- CONDENSE, statement 426
- CONDITION, ASSERT 928
- CONNECT, EXEC SQL 811
- CONSTANTS, statement 142
- constructor, instance constructor 177
- CONTEXT
 - DEMAND 1059
 - SUPPLY 1058
- CONTEXTS, statement, obsolete 1057
- CONTINUE, statement 307
- CONTROL, LOOP 550
- CONTROLS, statement 570
- CONVERT DATE
 - statement 921
 - statement, obsolete 1048
- CONVERT TEXT, statement 427
- CONVERT TIME STAMP, statement 919
- COPIES, print parameter, obsolete 1071
- CORRESPONDING FIELDS, SELECT 761
- cos, predefined function 110
- cosh, predefined function 110
- count(*), SELECT 751
- COUNT, SELECT 751
- COUNTRY
 - GET LOCALE 910
 - SET LOCALE 905
- COVER PAGE, print parameter, obsolete 1071
- COVER TEXT, print parameter, obsolete 1071
- CP, comparison operator 317
- CPI, PRINT-CONTROL 700
- CREATE DATA, statement 209
- CREATE OBJECT
 - OLE, statement 1006
 - statement 219
- CREATE PRIVATE, CLASS 159
- CREATE PROTECTED, CLASS 159
- CREATE PUBLIC, CLASS 159
- CS, comparison operator 317
- csequence, generic data type 97
- CUKY, data type 99
- CURR, data type 99
- CURRENCY, WRITE 659
- CURRENT LINE
 - MODIFY LINE 706
 - READ LINE 703
- CURRENT PAGE
 - MODIFY LINE 706
 - READ LINE 703
- CURRENT POSITION, TRUNCATE DATASET 877
- CURSOR
 - LOOP WITH CONTROL 552
 - LOOP, obsolete 1064
- cursor, data type 102
- CURSOR-SELECTION
 - FIELD MODULE 543
 - MODULE 539

D

- d, data type 93
- DATA
 - statement 131
 - statement, obsolete 1019
- DATA BUFFER
 - EXPORT 819
 - IMPORT 827
- DATA VALUES, INTERFACES 192
- data, generic data type 97
- DATABASE
 - DELETE 841
 - EXPORT 821
 - IMPORT 828
 - IMPORT DIRECTORY 839
- DATASET EXPIRATION, print parameter, obsolete 1071
- DATE
 - CONVERT 921
 - CONVERT TIME STAMP 919
 - CONVERT, obsolete 1048
- DATS, data type 99
- DAYLIGHT SAVING TIME
 - CONVERT DATE 921
 - CONVERT TIME STAMP 919
- dbmaxlen, predefined function 111
- DDMMYY, WRITE 666
- DEALLOCATE, COMMUNICATION 1081
- DEC, data type 99
- DECIMALS
 - ASSIGN CASTING 377
 - CREATE DATA 211
 - DATA 134
 - DESCRIBE FIELD 503
 - PARAMETERS 612
 - TYPES 123
 - WRITE 660
- DEFAULT
 - CALL FUNCTION STARTING NEW TASK 969
 - CLASS-EVENTS 191
 - EVENTS 189
 - FIELD-SYMBOLS 151
 - METHODS 171
 - OPEN DATASET 851
 - PARAMETERS 620
 - SELECTION-SCREEN TAB 597
 - SELECT-OPTIONS 634
 - SET CONNECTION 812
- DEFAULT KEY, TYPES 129
- DEFERRED
 - CLASS 165
 - INTERFACE 168
- DEFINE, statement 90
- DEFINING DATABASE, REPORT 57
- DEFINITION, CLASS 154
- DELETE
 - database table 799
 - database table, obsolete 1077
 - internal table 470
- DELETE DATASET, statement 879
- DELETE FROM, statement 841
- DELETING LEADING, SHIFT 423
- DELETING TRAILING, SHIFT 423
- DEMAND, statement, obsolete 1058
- DEPARTMENT, print parameter, obsolete 1071
- DESCENDING
 - SELECT 780
 - SORT – extract 492
 - SORT – internal table 476
 - SORT BY – extract 493
 - SORT BY – internal table 478
- DESCRIBE DISTANCE, statement 508
- DESCRIBE FIELD, statement 499
- DESCRIBE LIST, statement 711
- DESCRIBE TABLE, statement 506
- DESTINATION
 - CALL FUNCTION 967
 - CALL FUNCTION IN BACKGROUND TASK 973
 - CALL FUNCTION STARTING NEW TASK 968
 - print parameter, obsolete 1071
- DETAIL, statement, obsolete 1069
- DIRECTORY ENTRY
 - INSERT REPORT 952
 - SYNTAX-CHECK 945
- DISCONNECT, EXEC SQL 813
- DISPLAY
 - GET CURSOR – list 715
 - SET CURSOR – dynpro 568
 - SET CURSOR – list 719, 720
- DISPLAY LIKE, MESSAGE 736

DISPLAY-MODE
 EDITOR-CALL FOR REPORT 957
 EDITOR-CALL, obsolete 1086
 DISTINCT
 SELECT 748
 SELECT – aggregate 749
 DIV, arithmetic operator 395
 DIVIDE, statement 403
 DIVIDE-CORRESPONDING, statement,
 obsolete 1041
 DO, statement 334
 DUMMY, AUTHORITY-CHECK 890
 DURING LINE-SELECTION, TOP-OF-PAGE
 82
 DYNAMIC SELECTIONS, SELECTION-
 SCREEN 608

E

EDIT MASK
 DESCRIBE FIELD 505
 WRITE 664
 EDITOR-CALL FOR REPORT, statement 956
 EDITOR-CALL, statement, obsolete 1086
 ELSE, statement 332
 ELSEIF, statement 332
 ENCODING, OPEN DATASET 850
 END OF
 AT – extract 496
 AT – internal table 451
 AT SELECTION-SCREEN 79
 CLASS-DATA 141
 CONSTANTS 142
 DATA 137
 DATA, obsolete 1022
 SELECTION-SCREEN 584
 STATICS 144
 TYPES 126
 END OF FILE, SET DATASET 873
 END OF TASK, CALL FUNCTION STARTING
 NEW TASK 970
 ENDAT
 extract 496
 internal table 451
 ENDCASE, statement 333
 ENDCATCH, statement 355
 ENDCHAIN, dynpro statement 548
 ENDCLASS, statement 154
 ENDDO, statement 335
 ENDEXEC, statement 805
 ENDFORM, statement 65
 ENDFUNCTION, statement 64
 ENDIAN
 IMPORT 836
 OPEN DATASET 852
 ENDIF, statement 332
 ENDING AT
 CALL SCREEN 557
 CALL SELECTION-SCREEN 639
 SEARCH 414
 SEARCH – internal table 482
 WINDOW 721
 ENDINTERFACE, statement 167
 END-LINES, DESCRIBE LIST 712
 ENDLOOP
 ABAP statement 564
 dynpro statement 550
 extract 494
 internal table 447
 ENDMETHOD, statement 62
 ENDMODULE, statement 70
 END-OF-DEFINITION, statement 90
 END-OF-PAGE, statement 83
 END-OF-SELECTION, statement 75
 ENDON, statement, obsolete 1032
 ENDPROVIDE, statement 484
 ENDSELECT, statement 745
 ENDTRY, statement 350
 ENDWHILE, statement 337
 EQ
 relational operator 312
 WHERE 767
 error_message, CALL FUNCTION 277
 ERRORMESSAGE, FIELD, obsolete 1062
 ESCAPE, WHERE 768
 EVENT
 CLASS-METHODS 188
 METHODS 180
 EVENTS, statement 189
 EXCEPTIONS
 CALL FUNCTION 277
 CALL FUNCTION DESTINATION 967
 CALL METHOD 266
 CLASS-METHODS 185
 CREATE OBJECT 222
 METHODS 173
 RECEIVE RESULTS 972

- EXCEPTION-TABLE
 - CALL FUNCTION 280
 - CALL METHOD 272
 - CREATE OBJECT 222
- EXCLUDE, SELECTION-SCREEN 606
- EXCLUDING
 - GET PF-STATUS 560
 - SET PF-STATUS 559
- EXEC SQL, statement 805
- EXECUTE PROCEDURE, EXEC SQL 810
- EXISTS, subquery 774
- EXIT
 - loop 306
 - processing block 302
- EXIT FROM SQL, statement 815
- EXIT FROM STEP-LOOP, statement 576
- EXIT-COMMAND
 - AT SELECTION-SCREEN 82
 - MODULE 538
- exp, predefined function 110
- EXPONENT, WRITE 658
- EXPORT, statement 817
- EXPORTING
 - CALL DIALOG 1030
 - CALL FUNCTION 276
 - CALL FUNCTION DESTINATION 967
 - CALL METHOD 265
 - CALL METHOD OF – OLE 1007
 - CLASS-EVENTS 191
 - CLASS-METHODS 185
 - CREATE OBJECT 222
 - EVENTS 189
 - METHODS 170
 - RAISE EVENT 290
 - RAISE EXCEPTION 349
- EXPORTING LIST TO MEMORY, SUBMIT 242
- EXTRACT, statement 491
- F**
 - f, data type 93
 - FETCH
 - EXEC SQL 808
 - statement 783
 - FIELD
 - AUTHORITY-CHECK 890
 - dynpro statement 541
 - dynpro statement, obsolete 1060
 - GET CURSOR – dynpro 569
 - GET CURSOR – list 714
 - GET PARAMETER 898
 - GET RUN TIME 931
 - GET TIME 917
 - GET TIME STAMP 918
 - SET CURSOR – dynpro 568
 - SET CURSOR – list 718
 - SET PARAMETER 897
 - FIELD FORMAT, MODIFY LINE 706
 - FIELD MODULE, dynpro statement 541
 - FIELD SELECTION, SELECTION-SCREEN 607
 - FIELD VALUE
 - MODIFY LINE 706
 - READ LINE 704
 - FIELD-GROUPS, statement 152
 - FIELDS
 - ASSERT 928
 - GET node 74
 - PROVIDE 484
 - statement, obsolete 1025
 - FIELD-SYMBOLS, statement 150
 - FILTER, OPEN DATASET 856
 - FINAL
 - CLASS 158
 - METHODS 175
 - FINAL METHODS, INTERFACES 192
 - FIND, statement 409
 - FIRST
 - AT – extract 496
 - AT – internal table 451
 - FIRST OCCURRENCE, REPLACE 418
 - FIRST PAGE, SCROLL LIST 709
 - FIRST-LINE, DESCRIBE LIST 712
 - FIXED-POINT ARITHMETIC, INSERT REPORT 951
 - floor, predefined function 109
 - FLTP, data type 99
 - FONT, PRINT-CONTROL 700
 - FOR ALL ENTRIES, WHERE 776
 - FOR ALL INSTANCES, SET HANDLER 292
 - FOR EVENT
 - CLASS-METHODS 188
 - METHODS 180
 - FOR FIELD, SELECTION-SCREEN COMMENT 590
 - FOR SELECT, OPEN CURSOR 782

- FOR, SELECT-OPTIONS 629
- FORM, statement 65
- FORMAT, statement 674
- FORWARD, SCROLL LIST 709
- frac, predefined function 109
- FRAME, SELECTION-SCREEN 596
- FRAMES
 - FORMAT 679
 - WRITE 667
- FREE MEMORY, statement 842
- FREE OBJECT, OLE, statement 1012
- FREE SELECTIONS, SUBMIT 240
- FREE, statement 390
- FRIENDS, CLASS 161
- FROM
 - DELETE – database table 801
 - DELETE – internal table 471
 - INSERT 788
 - LOOP AT 449
 - MODIFY – database table 798
 - MODIFY – internal table 466
 - READ TABLE 437
 - SELECT 754
 - UPDATE 795
- FROM TABLE
 - DELETE 802
 - INSERT 789
 - MODIFY 799
 - REFRESH 1076
 - UPDATE 796
- FUNCTION
 - PRINT-CONTROL 700
 - statement 63
- FUNCTION KEY, SELECTION-SCREEN 600
- FUNCTION-POOL, statement 58

G

- GE
 - relational operator 312
 - WHERE 767
- GENERATE SUBROUTINE POOL, statement 937
- GET BIT, statement 433
- GET CONNECTION, EXEC SQL 812
- GET CURSOR
 - dynpro 569
 - list 714
- GET DATASET, statement 869

- GET LOCALE, statement 910
- GET node, statement 74
- GET PARAMETER, statement 898
- GET PF-STATUS, statement 560
- GET PROPERTY, OLE, statement 1010
- GET REFERENCE, statement 384
- GET RUN TIME, statement 931
- GET TIME STAMP, statement 918
- GET TIME, statement 917
- GIVING, ADD, obsolete 1036
- GLOBAL FRIENDS, CLASS 161
- GROUP, CALL FUNCTION STARTING NEW TASK 969
- GT
 - relational operator 312
 - WHERE 767

H

- HANDLE
 - ASSIGN CASTING 377
 - CREATE DATA 217
- HASHED TABLE
 - CREATE DATA 216
 - DATA 138
- hashed table, generic data type 97
- HASHED TABLE, TYPES 128
- HAVING, SELECT 778
- header
 - EXTRACT – field group 491
 - FIELD-GROUPS 152
 - INSERT – field group 489
- HEADER LINE, DATA 139
- HEAD-LINES, DESCRIBE LIST 712
- HELP-ID, DESCRIBE FIELD 504
- HELP-REQUEST
 - AT SELECTION-SCREEN 81
 - PARAMETERS 624
 - PROCESS 536
 - SELECT-OPTIONS 637
- HIDE, statement 697
- HIGH, SET RUN TIME CLOCK RESOLUTION 933
- HOLD
 - COMMUNICATION 1083
 - OPEN CURSOR 782
- HOTSPOT
 - FORMAT 677
 - WRITE 667

I

i, data type 93

ICON, WRITE 669

ID

ASSERT 928

AUTHORITY-CHECK 890

BREAK-POINT 930

COMMUNICATION 1079

DELETE 841

EXPORT 819

FREE MEMORY 842

GET PARAMETER 898

IMPORT 827

IMPORT DIRECTORY 839

MESSAGE 733

SELECTION-SCREEN 609

SET PARAMETER 897

IDS, SELECTION-SCREEN EXCLUDE 606

IF FOUND

INCLUDE 89

PERFORM 285

IF, statement 332

IGNORING CASE

FIND 411

REPLACE 419

IGNORING CONVERSION ERRORS

IMPORT 835

OPEN DATASET 858

IGNORING STRUCTURE BOUNDARIES,

IMPORT 834

IMMEDIATELY

print parameter, obsolete 1071

SET PF-STATUS 716

IMPORT DIRECTORY, statement 839

IMPORT, statement 825

IMPORTING

CALL DIALOG 1030

CALL FUNCTION 276

CALL FUNCTION DESTINATION 967

CALL METHOD 265

CLASS-METHODS 185

METHODS 170

RECEIVE RESULTS 972

IN

EXEC SQL 810

logical expression 325

REPLACE 418

subquery 774

WHERE – selection table 770

WHERE – value list 769

INCLUDE

GENERATE SUBROUTINE POOL 940

SELECTION-SCREEN 602

statement 89

SYNTAX-CHECK 946

INCLUDE STRUCTURE, statement 145

INCLUDE TYPE, statement 145

INCLUDING GAPS, PROVIDE 484

INCLUDING, CALL SUBSCREEN 554

INCREMENT, ASSIGN 375

INDEX

DELETE 473

DESCRIBE LIST 711

INSERT 458

MODIFY 467

MODIFY LINE 706

READ LINE 703

READ TABLE 440

SCROLL LIST 708

WRITE TO, obsolete 1052

index table, generic data type 97

INDEX TABLE, TYPES 128

INDEX-LINE, PRINT-CONTROL 701

INFOTYPES, statement, obsolete 1023

INHERITING FROM, CLASS 157

INIT DESTINATION, COMMUNICATION
1080

INITIAL

ASSIGN LOCAL COPY 1026

logical expression 322

INITIAL LINE

APPEND 462

INSERT 456

INITIAL SIZE

CREATE DATA 216

DATA 138

TYPES 130

INITIALIZATION, statement 72

INNER JOIN, SELECT 755

INOUT, EXEC SQL 810

INPUT

FIELD 543

FORMAT 677

MODULE 70

- OPEN DATASET 849
 - statement, obsolete 1069
- WRITE 667
- INSERT
 - database table 786
 - database table, obsolete 1077
 - field group 489
 - internal table 455
- INSERT REPORT, statement 948
- INSERT TEXTPOOL, statement 954
- INT1, data type 99
- INT2, data type 99
- INT4, data type 99
- INTENSIFIED
 - FORMAT 676
 - WRITE 667
- INTERFACE, statement 166
- INTERFACE-POOL, statement 59
- INTERFACES, statement 192
- INTERNAL TABLE
 - EXPORT 820
 - IMPORT 828
- INTO
 - FETCH 783
 - INSERT 489
 - LOOP AT 448
 - MESSAGE 738
 - READ TABLE 442
 - SELECT 760
- INVERSE
 - FORMAT 676
 - WRITE 667
- INVERTED-DATE, CONVERT, obsolete 1048
- IS INITIAL
 - CLASS-DATA 141
 - CONSTANTS 142
 - DATA 133
 - STATICS 143
- IS, logical expression 321

J

- JOB, SUBMIT 246
- JOIN, SELECT 755

K

- KEEP IN SPOOL, print parameter, obsolete 1071
- KEEPING DIRECTORY ENTRY, INSERT REPORT 950
- KEEPING TASK, RECEIVE RESULTS 971
- KEY
 - READ TABLE – database table, obsolete 1072
 - READ TABLE – internal table 439
 - READ TABLE – internal table, obsolete 1049
- KIND, DESCRIBE TABLE 506

L

- LANG, data type 99
- LANGUAGE
 - GET LOCALE 910
 - INSERT TEXTPOOL 954
 - READ TEXTPOOL 953
 - SET LOCALE 905
- LAST
 - AT – extract 496
 - AT – internal table 451
- LAST PAGE, SCROLL LIST 709
- LATE, GET node 75
- LAYOUT, print parameter, obsolete 1071
- LCHR, data type 99
- LE
 - relational operator 312
 - WHERE 767
- LEAVE LIST-PROCESSING, statement 724
- LEAVE PROGRAM, statement 299
- LEAVE SCREEN, statement 578
- LEAVE TO LIST-PROCESSING, statement 723
- LEAVE TO SCREEN, statement 579
- LEAVE TO TRANSACTION, statement 255
- LEAVE, statement, obsolete 1030
- LEFT
 - MOVE PERCENTAGE 1033
 - SCROLL LIST 709
 - SHIFT 423
- LEFT MARGIN, PRINT-CONTROL 701
- LEFT OUTER JOIN, SELECT 755
- LEFT-JUSTIFIED, WRITE 656

LEGACY BINARY MODE, OPEN DATASET 852
 LEGACY TEXT MODE, OPEN DATASET 853
 LENGTH
 COMMUNICATION 1083
 CREATE DATA 211
 DATA 134
 DESCRIBE FIELD 502
 FIND 410
 GET CURSOR – dynpro 570
 GET CURSOR – list 715
 PARAMETERS 610
 READ DATASET 868
 REPLACE 417, 1046
 TRANSFER 862
 TYPES 123
 LEVEL, PERFORM ON 289
 LIKE
 ASSIGN CASTING 377
 CLASS-DATA 141
 CONSTANTS 142
 CREATE DATA 210
 DATA 132
 PARAMETERS 612
 STATICS 143
 TYPES 122
 typing 199
 WHERE 768
 LINE
 DESCRIBE LIST 712
 GENERATE SUBROUTINE POOL 940
 GET CURSOR – dynpro 569
 GET CURSOR – list 714
 GET CURSOR FIELD – list 715
 MODIFY LINE 706
 PRINT-CONTROL 700
 READ LINE 703
 SCROLL LIST 709
 SELECTION-SCREEN 594
 SET CURSOR – dynpro 568
 SET CURSOR – list 718
 SET CURSOR FIELD – list 719
 SKIP 683
 SYNTAX-CHECK 945
 WRITE 671
 LINE FORMAT, MODIFY LINE 706
 LINE OF
 ASSIGN LOCAL COPY 1026
 CREATE DATA 213
 DATA 135
 TYPES 124
 typing 199
 LINE VALUE
 MODIFY LINE 706
 READ LINE 704
 LINE-COUNT
 DESCRIBE LIST 712
 NEW-PAGE 690
 print parameter, obsolete 1071
 REPORT 56
 SUBMIT 242
 LINES
 DESCRIBE LIST 712
 DESCRIBE TABLE 507
 LINES OF
 APPEND 463
 INSERT 456
 lines, predefined function 112
 LINES, RESERVE 696
 LINE-SIZE
 DESCRIBE LIST 712
 NEW-PAGE 691
 print parameter, obsolete 1071
 REPORT 56
 SUBMIT 242
 LIST AUTHORITY, print parameter, obsolete 1071
 LIST DATASET, print parameter, obsolete 1071
 LIST NAME, print parameter, obsolete 1071
 LISTBOX, PARAMETERS 618
 LITTLE ENDIAN, OPEN DATASET 852
 LOAD
 CLASS 165
 INTERFACE 168
 LOAD-OF-PROGRAM, statement 71
 LOCAL FRIENDS, CLASS 163
 LOCAL, statement, obsolete 1035
 log, predefined function 110
 log10, predefined function 110
 LOOP
 dynpro statement, obsolete 1064
 extract 494
 LOOP AT
 database table, obsolete 1074
 internal table 446

- LOOP AT SCREEN, statement 564
- LOOP WITH CONTROL, dynpro statement 550
- LOW, SET RUN TIME CLOCK RESOLUTION 933
- LOWER CASE
 - PARAMETERS 621
 - SELECT-OPTIONS 635
 - TRANSLATE 430
- LPI, PRINT-CONTROL 700
- LRAW, data type 99
- LT
 - relational operator 312
 - WHERE 767

M

- M, relational operator 320
- MAJOR-ID, IMPORT, obsolete 830
- MARK
 - SEARCH 415
 - SEARCH – internal table 483
- MATCHCODE OBJECT
 - PARAMETERS 621
 - SELECT-OPTIONS 636
- MAX, SELECT 750
- MAXIMUM LENGTH, READ DATASET 866
- MAXIMUM WIDTH
 - INSERT REPORT 948
 - READ REPORT 943
- MAXIMUM, statement, obsolete 1042
- me, self-reference 102
- MEMORY
 - DELETE 841
 - EXPORT 820
 - GET CURSOR 715
 - IMPORT 828
 - SET CURSOR 719, 720
- MEMORY ID
 - PARAMETERS 622
 - SELECT-OPTIONS 636
- MESSAGE
 - CALL FUNCTION DESTINATION 967
 - CALL FUNCTION STARTING NEW TASK 969
 - GENERATE SUBROUTINE POOL 939
 - OPEN DATASET 858
 - RECEIVE RESULTS 972

- statement 731
- SYNTAX-CHECK 945
- MESSAGE-ID
 - GENERATE SUBROUTINE POOL 941
 - REPORT 57
 - SYNTAX-CHECK 946
- MESSAGES INTO
 - CALL TRANSACTION 254
 - DEMAND 1059
- METHOD, statement 62
- METHODS, statement 169
- MIN, SELECT 750
- MINIMUM, statement, obsolete 1041
- MINOR-ID, IMPORT, obsolete 830
- MMDDYY, WRITE 666
- MOD, arithmetic operator 395
- MODE
 - CALL DIALOG 1029
 - CALL TRANSACTION 252
 - INFOTYPES 1023
- MODIF ID
 - PARAMETERS 619
 - SELECTION-SCREEN COMMENT 590
 - SELECTION-SCREEN PUSHBUTTON 592
 - SELECTION-SCREEN ULINE 589
 - SELECT-OPTIONS 634
- MODIFIER
 - GET LOCALE 910
 - SET LOCALE 905
- MODIFY
 - database table 797
 - database table, obsolete 1077
 - internal table 465
- MODIFY LINE, statement 705
- MODIFY SCREEN, statement 566
- MODULE
 - ABAP statement 70
 - dynpro statement 537
 - FIELD 543
- MOVE
 - statement 362
 - statement, obsolete 1033
- MOVE-CORRESPONDING, statement 364
- MULTIPLY, statement 402
- MULTIPLY-CORRESPONDING, statement, obsolete 1040

N

n

- data type 93
- generic data type 97
- NA, comparison operator 317
- NAME
 - GENERATE SUBROUTINE POOL 938
 - INFOTYPES 1023
- NE
 - relational operator 312
 - WHERE 767
- NESTING LEVEL, SELECTION-SCREEN 585
- NEW
 - AT – extract 496
 - AT – internal table 451
- NEW LIST IDENTIFICATION, print parameter, obsolete 1071
- NEW-LINE, statement 684
- NEW-PAGE
 - statement 688
 - statement, obsolete 1070
- NEW-SECTION
 - NEW-PAGE 693
 - statement, obsolete 1071
- NO DATABASE SELECTION, SELECT-OPTIONS 638
- NO DIALOG, NEW-PAGE 694
- NO END OF LINE, TRANSFER 863
- NO FIELDS, READ TABLE 442
- NO FLUSH
 - CALL METHOD OF – OLE 1007
 - CREATE OBJECT – OLE 1006
 - FREE OBJECT – OLE 1012
 - GET PROPERTY OF – OLE 1010
 - SET PROPERTY OF – OLE 1011
- NO INTERVALS
 - SELECTION-SCREEN BEGIN OF BLOCK 596
 - SELECTION-SCREEN BEGIN OF SCREEN 585
 - SELECT-OPTIONS 633
- NO STANDARD PAGE HEADING, REPORT 55
- NODE
 - PARAMETERS 624
 - SELECTION-SCREEN 609
- NODES, statement 147
- NO-DISPLAY
 - PARAMETERS 615
 - SELECT-OPTIONS 632
- NO-EXTENSION, SELECT-OPTIONS 633
- NO-GAP
 - ULINE 673
 - WRITE 657
- NO-GAPS, CONDENSE 426
- NO-GROUPING, WRITE 658
- NO-HEADING, NEW-PAGE 689
- NON-UNICODE, OPEN DATASET 851
- NON-UNIQUE KEY
 - CREATE DATA 216
 - DATA 138
 - TYPES 129
- NO-SCROLLING, NEW-LINE 684
- NO-SIGN, WRITE 659
- NOT
 - Boolean operator 328
 - WHERE 772
- NO-TITLE, NEW-PAGE 689
- NO-TOPOFPAGE, NEW-PAGE 692
- NO-ZERO, WRITE 659
- NP, comparison operator 317
- NS, comparison operator 317
- NULL
 - CLEAR 389
 - WHERE 770
- NUMBER FORMAT, TRANSLATE, obsolete 1045
- NUMBER OF PAGES, DESCRIBE LIST 712
- NUMBER, MESSAGE 733
- NUMC, data type 99
- numeric, generic data type 97
- numofchar, predefined function 111

O

- O, relational operator 320
- object, generic data type 97
- OBJECTS, CALL TRANSFORMATION 991
- OBLIGATORY
 - PARAMETERS 615
 - SELECT-OPTIONS 632
- OCCURS
 - DATA BEGIN OF, obsolete 1022
 - DATA, obsolete 1022
 - DESCRIBE TABLE 507
 - INFOTYPES 1023

- RANGES 1024
- TYPES, obsolete 1021
- OF, PERFORM 285
- OFFSET
 - FIND 410
- GENERATE SUBROUTINE POOL 941
- GET CURSOR – dynpro 570
- GET CURSOR – list 715
- REPLACE 417
- SET CURSOR – dynpro 568
- SET CURSOR – list 719, 720
- SYNTAX-CHECK 946
- ON
 - AT SELECTION-SCREEN 78
 - FIELD 543
 - MODULE 540
 - SELECT 755
- ON CHANGE OF, statement, obsolete 1031
- ON COMMIT, PERFORM 289
- ON ROLLBACK, PERFORM 289
- ONLY, OVERLAY 429
- OPEN CURSOR, statement 782
- OPEN DATASET, statement 847
- OPEN, EXEC SQL 808
- OPTION, SELECT-OPTIONS 634
- OPTIONAL
 - CLASS-EVENTS 191
 - EVENTS 189
 - METHODS 171
- OPTIONS FROM, CALL TRANSACTION 253
- OPTIONS, CALL TRANSFORMATION 991
- OR
 - Boolean operator 328
 - WHEN 333
 - WHERE 772
- ORDER BY, SELECT 779
- OTHERS
 - CALL FUNCTION 277
 - CALL FUNCTION DESTINATION 967
 - CALL FUNCTION STARTING NEW TASK 969
 - CALL METHOD 266
 - CATCH SYSTEM-EXCEPTIONS 355
 - RECEIVE RESULTS 972
 - WHEN 333
- OUT, EXEC SQL 810
- OUTPUT
 - AT SELECTION-SCREEN 78
- MODULE 70
- OPEN DATASET 849
- OUTPUT-LENGTH, DESCRIBE FIELD 503
- OVERLAY, statement 429

P

- p
 - data type 94
 - generic data type 98
- PACK, statement, obsolete 1034
- PACKAGE SIZE, SELECT 763
- PAGE
 - MODIFY LINE 706
 - READ LINE 703
 - SCROLL 709
 - SCROLL LIST 709
- PARAMETERS
 - CALL TRANSFORMATION 991
 - NEW-PAGE 694
 - SELECTION-SCREEN EXCLUDE 606
 - SELECTION-SCREEN INCLUDE 602
 - statement 610
- PARAMETER-TABLE
 - CALL FUNCTION 279
 - CALL METHOD 271
 - CREATE OBJECT 222
- PERCENTAGE, MOVE, obsolete 1033
- PERFORM, statement 284
- PERFORMING
 - CALL FUNCTION STARTING NEW TASK 970
 - EXEC SQL 814
- PERSON TABLE, INFOTYPES 1023
- PLACES
 - SCROLL LIST 709
 - SHIFT 421
- POSITION
 - GET DATASET 869
 - OPEN DATASET 854
 - PRINT-CONTROL 700
 - SELECTION-SCREEN 594
 - SET DATASET 873
 - statement 686
 - TRUNCATE DATASET 877
- PREC, data type 99
- PREFERRED PARAMETER
 - CLASS-METHODS 185
 - METHODS 171

- PRIMARY KEY, SELECT 780
- PRINT OFF, NEW-PAGE 694
- PRINT ON
 - NEW-PAGE 693
 - NEW-PAGE, obsolete 1070
- PRINT-CONTROL, statement 699
- PRIVATE SECTION, statement 155
- PROCESS, dynpro statement 536
- PROGRAM
 - PERFORM 285
 - SET PF-STATUS 559
 - SET TITLEBAR 561
 - statement 57
 - SYNTAX-CHECK 945
- PROGRAM TYPE, INSERT REPORT 950
- PROTECTED SECTION, statement 155
- PROVIDE
 - statement 483
 - statement, obsolete 1054
- PUBLIC
 - CLASS 157
 - CLASS DEFERRED 165
 - INTERFACE 167
- PUBLIC SECTION, statement 155
- PUSHBUTTON
 - SELECTION-SCREEN 592
 - SELECTION-SCREEN INCLUDE 604
- PUT, statement 295

Q

- QUAN, data type 99
- QUEUE-ONLY
 - CALL METHOD OF – OLE 1007
 - CREATE OBJECT – OLE 1006
 - GET PROPERTY OF – OLE 1010
- QUICKINFO, WRITE 672

R

- RADIOBUTTON GROUP
 - AT SELECTION-SCREEN 79
 - PARAMETERS 617
- RADIOBUTTON GROUPS, SELECTION-SCREEN EXCLUDE 606
- RAISE EVENT, statement 290
- RAISE EXCEPTION, statement 349
- RAISE, statement 357
- RAISING
 - CLASS-METHODS 185

- FORM 68
- MESSAGE 737
- METHODS 172
- RANGE
 - ADD, obsolete 1036
 - ASSIGN 382
 - DO 335
 - WHILE 338
- RANGE OF
 - DATA 140
 - TYPES 131
- RANGES, statement, obsolete 1024
- RAW, data type 99
- RAWSTRING, data type 99
- READ DATASET, statement 864
- READ LINE, statement 703
- READ REPORT, statement 943
- READ TABLE
 - database table, obsolete 1072
 - internal table 436
 - internal table, obsolete 1049
- READ TEXTPOOL, statement 953
- READ-ONLY
 - CLASS-DATA 141
 - DATA 134
- RECEIVE BUFFER, COMMUNICATION 1081
- RECEIVE RESULTS, statement 971
- RECEIVED, COMMUNICATION 1083
- RECEIVER, print parameter, obsolete 1071
- RECEIVING, CALL METHOD 265
- REDEFINITION, METHODS 181
- REF TO
 - CREATE DATA 215
 - DATA 136
 - TYPES 124
 - typing 199
- REFERENCE INTO
 - APPEND 465
 - COLLECT 461
 - INSERT 459
 - LOOP AT 448
 - MODIFY 469
 - READ TABLE 445
- REFERENCE, METHODS 171
- REFRESH
 - statement 389
 - statement, obsolete 1075

- REFRESH CONTROL, statement 576
- REJECT, statement 304
- RENAMING
 - INCLUDE STRUCTURE 145
 - INCLUDE TYPE 145
- REPLACE
 - statement 416
 - statement, obsolete 1046
- REPLACEMENT CHARACTER
 - IMPORT 835
 - OPEN DATASET 859
- REPLACEMENT COUNT, REPLACE 419
- REPLACEMENT LENGTH, REPLACE 419
- REPLACEMENT OFFSET, REPLACE 419
- REPORT, statement 54
- REQUEST, FIELD 544
- REQUESTED, logical expression 323
- RESERVE, statement 696
- RESET
 - FORMAT 680
 - WRITE 667
- RESPECTING CASE
 - FIND 411
 - REPLACE 419
- RESULT, CALL TRANSFORMATION 991
- RETURN
 - statement 301
 - SUBMIT 230
- RETURN TO SCREEN, LEAVE TO LIST-PROCESSING 723
- RETURNCODE, COMMUNICATION 1082
- RETURNING
 - CLASS-METHODS 185
 - METHODS 176
- RIGHT
 - MOVE PERCENTAGE 1033
 - SCROLL LIST 709
 - SHIFT 423
- RIGHT-JUSTIFIED, WRITE 656
- ROLLBACK WORK, statement 886
- ROUND, WRITE 661

S

- s, data type 94
- SAP COVER PAGE, print parameter,
 - obsolete 1071
- SAP-SPOOL
 - SUBMIT 244

- SUBMIT, obsolete 1070
- SCREEN
 - REFRESH CONTROL 576
 - SELECTION-SCREEN 584
- screen
 - LOOP AT SCREEN 564
 - MODIFY SCREEN 566
- screen, structure 108
- SCROLL LIST, statement 708
- SCROLLING, NEW-LINE 684
- SEARCH
 - byte and character strings 412
 - internal table 481
 - READ TABLE, obsolete 1072
- SEARCH PATTERN, PARAMETERS 625
- SECONDS
 - WAIT UNTIL 972
 - WAIT UP TO 338
- SECTION OF
 - FIND 410
 - REPLACE 417
- SELECT
 - FIELD, obsolete 1062
 - statement 745
 - statement, obsolete 1077
 - subquery 773
- SELECTION-SCREEN
 - statement 582
 - SUBMIT 233
- SELECTION-SET
 - CALL SELECTION-SCREEN 639
 - SUBMIT 235
- SELECTION-SETS, SUBMIT 235
- SELECTION-TABLE, SUBMIT 236
- SELECT-OPTIONS
 - CHECK 303
 - SELECTION-SCREEN EXCLUDE 606
 - SELECTION-SCREEN INCLUDE 603
 - statement 626
- SEND BUFFER, COMMUNICATION 1081
- sender
 - CLASS-METHODS 188
 - event parameter 190
 - METHODS 180
- SEPARATE UNIT, CALL FUNCTION IN BACKGROUND TASK 973
- SEPARATED BY, CONCATENATE 408
- SET BIT, statement 432

- SET BLANK LINES, statement 681
- SET CONNECTION, EXEC SQL 812
- SET COUNTRY, statement 911
- SET CURSOR
 - dynpro 567
 - list 718
- SET DATASET, statement 872
- SET EXTENDED CHECK, statement 935
- SET HANDLER, statement 291
- SET HOLD DATA, statement 577
- SET LANGUAGE, statement 904
- SET LEFT SCROLL-BOUNDARY, statement 687
- SET LOCALE, statement 905
- SET MARGIN, statement 698
- SET PARAMETER, statement 896
- SET PF-STATUS
 - dynpro 558
 - list 716
- SET PROPERTY, OLE, statement 1011
- SET RUN TIME ANALYZER, statement 934
- SET RUN TIME CLOCK RESOLUTION, statement 933
- SET SCREEN, statement 578
- SET TITLEBAR
 - dynpro 561
 - list 718
- SET UPDATE TASK LOCAL, statement 887
- SET USER-COMMAND, statement 297
- SET, UPDATE 792
- SHARED BUFFER
 - DELETE 841
 - EXPORT 823
 - IMPORT 830
- SHARED MEMORY
 - DELETE 841
 - EXPORT 823
 - IMPORT 830
- SHARED MEMORY ENABLED, CLASS 160
- SHIFT, statement 420
- SHORTDUMP-ID, GENERATE
 - SUBROUTINE POOL 942
- sign, predefined function 109
- SIGN, SELECT-OPTIONS 634
- simple, generic data type 98
- sin, predefined function 110
- SINGLE, SELECT 748
- sinh, predefined function 110
- SIZE, PRINT-CONTROL 701
- SKIP
 - SELECTION-SCREEN 588
 - statement 682
- SKIP FIRST SCREEN
 - CALL DIALOG 1029
 - CALL TRANSACTION 250
 - LEAVE TO TRANSACTION 256
- SOME, subquery 775
- SORT
 - extract 492
 - internal table 476
- SORTABLE CODE, CONVERT TEXT 427
- SORTED BY, APPEND 463
- SORTED TABLE
 - CREATE DATA 216
 - DATA 138
- sorted table, generic data type 98
- SORTED TABLE, TYPES 128
- SOURCE, CALL TRANSFORMATION 991
- space, constant 102
- SPLIT, statement 424
- SPOOL DYNPRO, SUBMIT 244
- SPOOL PARAMETERS, SUBMIT 244
- sqrt, predefined function 110
- SSTRING, data type 99
- STABLE
 - SORT – extract 492
 - SORT – internal table 476
- STANDARD TABLE
 - CREATE DATA 216
 - DATA 138
- standard table, generic data type 98
- STANDARD TABLE, TYPES 128
- STARTING AT
 - CALL SCREEN 557
 - CALL SELECTION-SCREEN 639
 - SEARCH 414
 - SEARCH – internal table 482
 - WINDOW 721
- STARTING NEW TASK, CALL FUNCTION 968
- START-OF-SELECTION, statement 73
- STATICS, statement 143
- STATUSINFO, COMMUNICATION 1081
- STOP, statement 305
- STRING, data type 99
- string, data type 94

- strlen, predefined function 111
- STRUCTURE
 - ASSIGN COMPONENT OF 375
 - FIELD-SYMBOLS 151
 - FORM 67
 - INCLUDE 145
- SUBKEY, ASSERT 928
- SUBMIT
 - statement 230
 - statement, obsolete 1070
- SUBSCREEN
 - CALL 554
 - SELECTION-SCREEN 585
- SUBSTRING, REPLACE 418
- SUBTRACT, statement 402
- SUBTRACT-CORRESPONDING, statement, obsolete 1039
- SUM
 - SELECT 750
 - statement 453
- sum, AT 497
- SUMMARY, statement, obsolete 1069
- SUMMING, statement, obsolete 1042
- super->
 - METHODS constructor 178
 - METHODS REDEFINITION 182
- SUPPLIED, logical expression 324
- SUPPLY, statement, obsolete 1058
- SUPPRESS DIALOG, statement 563
- sy, system fields 102
- SYMBOL, WRITE 670
- SYNTAX-CHECK, statement 944
- SYST, system fields 102

T

- t, data type 94
- TAB, SELECTION-SCREEN 597
- TABBED BLOCK, SELECTION-SCREEN 597
- TABLE
 - INSERT 457
 - MODIFY 467
 - PARAMETERS 624
 - SELECT INTO 763
 - SELECTION-SCREEN 609
 - SPLIT 424
- TABLE FIELD, ASSIGN 372
- TABLE KEY
 - DELETE 472

- READ TABLE 437
- TABLE OF
 - CREATE DATA 216
 - DATA 138
 - TYPES 127
- TABLE, DELETE 471
- table, generic data type 98
- table_line
 - pseudo component 480
 - table key 129
- TABLES
 - CALL FUNCTION 276
 - CALL FUNCTION DESTINATION 967
 - FORM 69
 - PERFORM 287
 - RECEIVE RESULTS 972
 - statement 146
- TABLES *, statement, obsolete 1020
- TABLEVIEW, CONTROLS 571
- TABSTRIP, CONTROLS 571
- tan, predefined function 110
- tanh, predefined function 110
- TESTING
 - CLASS 161
 - METHODS 183
- TEXT
 - SORT – extract 492
 - SORT – internal table 476
 - SORT BY – extract 493
 - SORT BY – internal table 478
- TEXT MODE, OPEN DATASET 850
- TEXTPOOL
 - INSERT 954
 - READ 953
- THEN, ADD, obsolete 1036
- TIME
 - CONVERT TIME STAMP 919
 - GET 917
- TIME STAMP
 - CONVERT 919
 - CONVERT DATE 921
 - GET 918
- TIME ZONE
 - CONVERT DATE 921
 - CONVERT TIME STAMP 919
 - WRITE 663
- TIMES, DO 335
- TIMS, data type 99

TITLE

- EDITOR-CALL, obsolete 1086
- SELECTION-SCREEN BEGIN OF BLOCK 596
- SELECTION-SCREEN BEGIN OF SCREEN 584
- TITLE-LINES, DESCRIBE LIST 712
- TO, LOOP AT 449
- TOP-LINES, DESCRIBE LIST 712
- TOP-OF-PAGE, statement 82
- TRANSFER, statement 860
- TRANSLATE
 - statement 430
 - statement, obsolete 1044
- TRANSPORTING
 - MODIFY 468
 - READ TABLE 442
- TRANSPORTING NO FIELDS
 - LOOP AT 448
 - READ TABLE 446
- trunc, predefined function 109
- TRUNCATE DATASET, statement 877
- TRY, statement 350
- TYPE
 - ASSIGN CASTING 377
 - ASSIGN, obsolete 379
 - CLASS-DATA 141
 - CONSTANTS 142
 - CONTROLS 571
 - CREATE DATA 210
 - CREATE OBJECT 219
 - DATA 132
 - DESCRIBE FIELD 500
 - INCLUDE 145
 - MESSAGE 731
 - MESSAGE – text 735
 - NODES 147
 - OPEN DATASET 856
 - PARAMETERS 612
 - RAISE EXCEPTION 349
 - STATICS 143
 - TYPES 122
 - typing 199
- TYPE-POOL, statement 59
- TYPE-POOLS, statement 121
- TYPES, statement 122

U

- ULINE
 - SELECTION-SCREEN 589
 - statement 673
- UNASSIGN, statement 383
- UNDER, WRITE 657
- UNICODE ENABLING, INSERT REPORT 951
- UNIQUE KEY
 - CREATE DATA 216
 - DATA 138
 - TYPES 129
- UNIT
 - data type 99
 - WRITE 662
- UNPACK, statement 367
- UNTIL, ADD, obsolete 1036
- UP TO
 - SELECT 759
 - SHIFT 422
 - WAIT UNTIL 972
- UPDATE
 - CALL TRANSACTION 252
 - database table 790
 - database table, obsolete 1077
 - OPEN DATASET 849
 - SELECT 748
- UPDATE TASK, CALL FUNCTION 283
- UPPER CASE, TRANSLATE 430
- USER, SUBMIT 246
- USER-COMMAND
 - PARAMETERS AS CHECKBOX 616
 - PARAMETERS AS LISTBOX 618
 - PARAMETERS RADIOBUTTON GROUP 617
 - SELECTION-SCREEN PUSHBUTTON 592
 - SELECTION-SCREEN TAB 597
- USING
 - CALL DIALOG 1029
 - CALL SELECTION-SCREEN 639
 - CALL TRANSACTION 250
 - FORM 66
 - PERFORM 288
 - TRANSLATE 431
 - WRITE 664
- USING SCREEN, CONTROLS 571
- UTF-8, OPEN DATASET 851

V

VALID BETWEEN, DATA, obsolete 1022
VALID FROM TO, INFOTYPES 1023
VALUE
 CLASS-DATA 141
 CLASS-EVENTS 191
 CONSTANTS 142
 DATA 133
 EVENTS 189
 FORM 66
 GET CURSOR – dynpro 570
 GET CURSOR – list 715
 METHODS 171
 STATICS 143
VALUE CHECK, PARAMETERS 623
VALUE-REQUEST
 AT SELECTION-SCREEN 81
 PARAMETERS 625
 PROCESS 536
 SELECT-OPTIONS 637
VALUES
 FIELD, obsolete 1061
 INSERT 788
VARY, WHILE 338
VARYING, DO 335
VERSION
 DELETE, obsolete 1077
 LOOP, obsolete 1074
 MODIFY, obsolete 1077
 READ TABLE, obsolete 1072
 SELECTION-SCREEN 606
VISIBLE LENGTH
 PARAMETERS 616
 PARAMETERS AS LISTBOX 618
 SELECTION-SCREEN COMMENT 590
 SELECTION-SCREEN PUSHBUTTON 592
 SELECT-OPTIONS 633

W

WAIT UNTIL, statement 972
WAIT UP TO, statement 338
WAIT, COMMIT WORK 885
WARNING, FIELD, obsolete 1062
WHEN, statement 333
WHENEVER FOUND, FIELD, obsolete 1062
WHERE

DELETE – database table 800
DELETE – internal table 474
LOOP AT 449
MODIFY – internal table 469
PROVIDE 484
SELECT 765
UPDATE 792
WHILE, statement 337
WINDOW
 SELECTION-SCREEN BEGIN OF 584
 statement 721
WITH
 AT 496
 CLEAR 388
 FIELD 547
 MESSAGE 739
 SET TITLEBAR 561
 SUBMIT 234
WITH-HEADING, NEW-PAGE 689
WITH-TITLE, NEW-PAGE 689
WORD
 GENERATE SUBROUTINE POOL 940
 SYNTAX-CHECK 945
WRITE /, statement 684
WRITE TO
 internal table, obsolete 1052
 statement 366
WRITE, statement 647

X

x
 data type 94
 generic data type 98
XML
 CALL TRANSFORMATION RESULT 994
 CALL TRANSFORMATION SOURCE 992
xsequence, generic data type 98
xstring, data type 94
xstrlen, predefined function 111

Y

YYMMDD, WRITE 666

Z

Z, relational operator 320