Open Group Technical Standard

DRDA Volume 2:

Formatted Data Object Content Architecture (FD:OCA)

The Open Group

Open Group Technical Standard

DRDA Volume 2: Formatted Data Object Content Architecture (FD:OCA)

ISBN: 1-85912-201-9
Document Number: C813

Published in the U.K. by The Open Group, October 1998.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

Contents

List of Figures

List of Tables

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the *IT DialTone*. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world's largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- Consolidating, prioritizing, and communicating customer requirements to vendors
- Conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- Managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- Adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- Licensing and promoting the Open Brand, represented by the ''X'' Device, that designates vendor products which conform to Open Group Product Standards
- Promoting the benefits of the IT DialTone to customers, vendors, and the public

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development, and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trademark on behalf of the industry.

Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of Technical Standards (formerly CAE and Preliminary Specifications) through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product.

The ''X'' Device is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the Open Brand Trade Mark License Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical Standards and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys, and business titles.

There are several types of specification:

- *Technical Standards* (formerly *CAE Specifications*)

  The Open Group Technical Standards form the basis for our Product Standards. These Standards are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement a Technical Standard can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. Technical Standards are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *CAE Specifications*

  CAE Specifications and Developers' Specifications published prior to January 1998 have the same status as Technical Standards (see above).

- *Preliminary Specifications*

  Preliminary Specifications have usually addressed an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is as stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

  Preliminary Specifications are analogous to the *trial-use* standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a Technical Standard. While the intent is to progress Preliminary Specifications to corresponding Technical Standards, the ability to do so depends on consensus among Open Group members.

- *Consortium and Technology Specifications*

  The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

  Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as Technical Standards, in which case the relevant Technology Specification is superseded by a Technical Standard.

In addition, The Open Group publishes:

- *Product Documentation*

  This includes product documentation—programmer's guides, user manuals, and so on—relating to the Pre-structured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

- *Guides*

  These provide information that is useful in the evaluation, procurement, development, or management of open systems, particularly those that relate to the Technical Standards or Preliminary Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

- *Technical Studies*

  Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.

- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at *http://www.opengroup.org/corrigenda*.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at *http://www.opengroup.org/pubs*.

This Document

The *Distributed Relational Database Architecture Specification* comprises three volumes:

- *Distributed Relational Database Architecture (DRDA)* (the DRDA Reference)
- *Formatted Data Object Content Architecture (FD:OCA)* (the FD:OCA Reference)
- *Distributed Data Management (DDM) Architecture* (the DDM Reference)

This volume, *Formatted Data Object Content Architecture*, describes the functions and services that make up the Formatted Data Object Content Architecture (FD:OCA). This architecture makes it possible to bridge the connectivity gap between environments with different data types and data representations methods.

The FD:OCA is embedded in the Distributed Relational Database Architecture (DRDA), which identifies and brackets the Formatted Data Object in its syntax. DRDA describes the connectivity between relational database managers that enables applications programs to access distributed relational data.

This book is divided into five chapters:

- Chapter 1 on page 1 introduces the DRDA specification set.
- Chapter 2 on page 5 briefly states the requirements, purpose, objectives, and functions of FD:OCA.
- Chapter 3 on page 11 introduces the concepts that form the basis of FD:OCA.
- Chapter 4 on page 25 provides the syntax, semantics, and pragmatics of the data structures found in FD:OCA.
- Chapter 5 on page 73 describes functional subsets and towers within FD:OCA, and defines what it means to be in compliance with the architecture.

The *Glossary* defines terms used within the book.

Intended Audience

This volume is intended as a reference for systems programmers and other developers who need to develop or adapt a product or program to attach to a communications network. Specifically, it will be used when developing implementations of the Distributed Relational Database Architecture.

This book is a reference, not a tutorial. It is intended to complement individual product publications, but not to describe product implementations of the architecture.

Typographic Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used for system elements that must be used literally, such as interface names and defined constants.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote function names and variable values such as interface arguments.
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.

- The notation [EABCD] is used to identify an error value EABCD.
- Syntax, code examples, and user input in interactive examples are shown in fixed width font.
- Variables within syntax statements are shown in *italic fixed width font*.

Problem Reporting

For any problems with DRDA-based software or vendor-supplied documentation, contact the software vendor's customer service department. Comments relating to this Open Group Technical Standard, however, should be sent to the addresses provided on the copyright page.

Trademarks

Motif®, OSF/1®, UNIX®, and the ''X Device''® are registered trademarks and IT DialTone™ and The Open Group™ are trademarks of The Open Group in the U.S. and other countries.

HP-UX® is a registered trademark of Hewlett-Packard Company.

The following are trademarks of the IBM Corporation in the United States and other countries:

AIX®
AS/400®
DATABASE 2®
DB2®
Distributed Relational Database Architecture®
DRDA®
IBM®
MVS®
Netview®
OS/2®
OS/390®
OS/400®
RISC System/6000®
SQL/DS®
System/390®
VM®

Intel® is a registered trademark of Intel Corporation.

Microsoft® and Windows NT® are registered trademarks of Microsoft Corporation.

NFS® is a registered trademark and Network File System™ is a trademark of Sun Microsystems, Inc.

Solaris® is a registered trademark of Sun Microsystems, Inc.

VAX® is a registered trademark of Digital Equipment Corporation.

Referenced Documents

These publications provide the background for understanding DRDA.

DRDA Overview

For an overview of DRDA, read:

- Open Group Technical Standard, TBD 1998, DRDA Volume 1: Distributed Relational Database Architecture (DRDA) (ISBN: 1-85912-295-7, C812).

The DRDA Processing Model and Command Flows

These publications help the reader to understand the DDM documentation and what is needed to implement the base functions required for a DRDA product:

- Open Group Technical Standard, TBD 1998, DRDA Volume 3: Distributed Data Management (DDM) Architecture (ISBN: 1-85912-206-X, C814).
- *Distributed Data Management Architecture General Information*, GC21-9527 (IBM).
- *Distributed Data Management Architecture Implementation Programmer's Guide*, SC21-9529 (IBM).
- *Character Data Representation Architecture Reference*, SC09-1390 (IBM).
- *Character Data Representation Architecture Registry*, SC09-1391 (IBM).

Communications, Security, Accounting, and Transaction Processing

For information about distributed transaction processing, see the following:

- CAE Specification, November 1995, Distributed Transaction Processing: The CPI-C Specification, Version 2 (ISBN: 1-85912-135-7, C419), published by The Open Group.

The following publications contain background information adequate for an in-depth understanding of DRDA's use of TCP/IP:

- *Internetworking With TCP/IP Volume I: Principles, Protocols, and Architecture*, Douglas E. Corner, Prentice Hall, Englewood Cliffs, New Jersey, 1991, SC31-6144 (IBM).
- *Internetworking With TCP/IP Volume II: Implementation and Internals*, Douglas E. Corner, Prentice Hall, Englewood Cliffs, New Jersey, 1991, SC31-6145 (IBM).
- *Internetworking With TCP/IP*, Douglas E. Corner, SC09-1302 (IBM).
- *UNIX Network Programming*, W. Richard Stevens, Prentice Hall, Englewood Cliffs, New Jersey, 1990, SC31-7193 (IBM).
- *UNIX Networking*, Kochan and Wood, Hayden Books, Indiana, 1989.
- *Introduction to IBM's Transmission Control Protocol/Internet Protocol Products for OS/2, VM, and MVS*, GC31-6080 (IBM).
- *Transmission Control Protocol*, RFC 793, Defense Advanced Research Projects Agency (DARPA).

Many IBM publications contain detailed discussions of SNA concepts and the LU 6.2 architecture. The following publications contain background information adequate for an in-depth understanding of DRDA's use of LU 6.2 functions:

- *SNA Concepts and Products*, GC30-3072 (IBM).
- *SNA Technical Overview*, GC30-3073 (IBM).
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2,* GC30-3084 (IBM).
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808 (IBM).
- *SNA Management Services: Alert Implementation Guide*, SCnn-nnnn (IBM).
- *SNA Format and Protocol Reference Manual: Architecture Logic For LU Type 6.2* SC30-3269 (IBM).

These are publications that contain background for DRDA's use of The Open Group's OSF DCE security. A listing of security publications is available on The Open Group website at *http://www.opengroup.org*, under publications. Many titles are available for browsing in HTML.

- CAE Specification, December 1995, Generic Security Service API (GSS-API) Base (ISBN: 1-85912-131-4, C441), published by The Open Group.
- CAE Specification, August 1997, DCE 1.1: Authentication and Security Services (C311), published by The Open Group as an electronic document (via ftp).
- *The Open Group's OSF DCE SIG Request For Comments 5.x*, GSS-API Extensions for DCE, available from The Open Group.
- *IETF Request For Comments 1508*, Generic Security Service Application Program Interface.
- *IETF Request For Comments 1510*, The Kerberos Network Authentication Service (V5).

Data Definition and Exchange

The following publications describe ISO SQL, FD:OCA, and CDRA:

- Open Group Technical Standard, TBD 1998, DRDA Volume 2: Formatted Data Object Content Architecture (FD:OCA) (ISBN: 1-85912-201-9, C813) (this document).
- ISO/IEC 9075: 1992, Information Technology — Database Language SQL (technically identical to ANSI standard X3.135-1992).
- *Character Data Representation Architecture Reference*, SC09-1390 (IBM).
- *Character Data Representation Architecture Registry*, SC09-1391 (IBM).
- *Character Data Representation Architecture, Executive Overview*, GC09-1392 (IBM).

Other

- *ANSI/IEEE Std. 745-1985, Binary Floating Point Arithmetic*.
- *Dictionary of Computing—Information Processing, Personal Computing, Telecommunications, Print and Office Systems, IBM-Specific Terms*, SC20-1699 (IBM).

Chapter 1

The DRDA Specification

The *Distributed Relational Database Architecture Specification* comprises three volumes:

- *Distributed Relational Database Architecture (DRDA)* (the DRDA Reference)
- *Formatted Data Object Content Architecture (FD:OCA)* (the FD:OCA Reference)
- *Distributed Data Management (DDM) Architecture* (the DDM Reference)

DRDA is an open, published architecture that enables communication between applications and database systems on disparate platforms, whether those applications and database systems are provided by the same or different vendors and whether the platforms are the same or different hardware/software architectures. DRDA is a combination of other architectures and the environmental rules and process model for using them. The architectures that actually comprise DRDA are Distributed Data Management (DDM) and Formatted Data Object Content Architecture (FD:OCA).

The Distributed Data Management (DDM) architecture provides the overall command and reply structure used by the distributed database. Fewer than 20 commands are required to implement all of the distributed database functions for communication between the Application Requester (client) and the Application Server.

The Formatted Data Object Content Architecture (FD:OCA) provides the data definition architectural base for DRDA. Descriptors defined by DRDA provide layout and data type information for all the information routinely exchanged between the Application Requesters and Servers. A descriptor organization is defined by DRDA to allow dynamic definition of user data that flows as part of command or reply data. DRDA also specifies that the descriptors only have to flow once per answer set, regardless of the number of rows actually returned, thus minimizing data traffic on the wire.

It is recommended that the DRDA Reference be used as the main source of information and roadmap for implementing DRDA. This section describes the relationships among the above three volumes and provides the details on how they are used to develop a DRDA requester (client) or server. Overviews of DRDA and DDM are provided in this section and in more detail in the introductory sections of their respective volumes.

It is recommended that the introductory chapter of the DDM Reference, which describes its overall structure and basic concepts, is read either before reading the chapter in the DRDA Reference entitled The DRDA Processing Model and Command Flows or in conjunction with it. The rest of the DDM Reference should be used primarily as a reference when additional detail is needed to implement the functions and flows as defined in the DRDA Reference.

DRDA can flow over either SNA or TCP/IP transport protocols and the details and differences in doing so are provided in the third part of the DRDA Reference. It is expected that the developer is familiar with whichever transport protocol will be supported, as that level of detail is not provided in this documentation. Even if only implementing for TCP/IP, it is recommended that the developer be familiar with the two-phase commit recovery model as described in SNA LU 6.2 since that is the model used by DRDA for either of the transport protocols.

Besides SNA and TCP/IP, DRDA also uses the following other architectures:

- Character Data Representation Architecture (CDRA)
- SNA Management Services Architecture (MSA) for problem determination support
- The Open Group Distributed Computing Environment (DCE)

For a better understanding of DRDA, the reader should have some familiarity with these architectures. (See **Referenced Documents** on page xi.)

Finally, DRDA is based on the Structured Query Language (SQL) but is not dependent on any particular level or dialect of it. It is not necessary to know the details of how to construct all the SQL statements, only to recognize certain types of statements and any host variables they may contain in order to map them to their DRDA equivalents.

1.1 The DRDA Reference

The DRDA Reference describes the necessary connection between an application and a relational database management system in a distributed environment. It describes the responsibilities of these participants, and specifies when the flows should occur. It describes the formats and protocols required for distributed database management system processing. It does *not* describe an Application Programming Interface (API) for distributed database management system processing.

This reference is divided into three parts. The first part describes the database access protocols. The second part describes the environmental support that DRDA requires, which includes network support. The third part contains the specific network protocols and characteristics of the environments these protocols run in, along with how these network protocols relate to DRDA.

1.2 The FD:OCA Reference

The FD:OCA Reference describes the functions and services that make up the Formatted Data Object Content Architecture (FD:OCA). This architecture makes it possible to bridge the connectivity gap between environments with different data types and data representation methods by providing constructs that describe the data being exchanged between systems.

The FD:OCA is embedded in the Distributed Relational Database Architecture, which identifies and brackets the Formatted Data Object in its syntax. DRDA describes the connectivity between relational database managers that enables applications programs to access distributed relational data and uses FD:OCA to describe the data being sent to the server and/or returned to the requester. For example, when data is being sent to the server for inserting into the database or being returned to the requester as a result of a database query, the data type (character, integer, floating point, and so on) and its characteristics (length, precision, byte-reversed or not, and so on) are all described by FD:OCA.

The FD:OCA Reference is presented in three parts:

- Overview material to give the reader a feel for FD:OCA. This material can be skimmed.
- Example material that shows how the FD:OCA mechanisms are used. This should be read for understanding.
- References to the detailed FD:OCA descriptions. A few of these topics should be read up front to gain experience with the style of presentation and the content of the first several

triplets. The rest can be read when the level of detail presented in that chapter is required. This is reference material.

1.3 The DDM Reference

The DDM Reference describes the architected commands, parameters, objects, and messages of the DDM data stream. This data stream accomplishes the data interchange between the various pieces of the DDM model.

Chapter 2

Introduction to FD:OCA

This chapter:

- Outlines the requirements for FD:OCA
- Describes how FD:OCA meets these requirements
- Illustrates the applicability of FD:OCA with an example
- Explains how to use this volume

2.1 Why is FD:OCA Needed

In the world of distributed and network computing, it is necessary to interchange electronic objects of various kinds, with and among like and unlike environments. This includes interchange with host computers, where the traditional files and databases are kept—the information assets of an enterprise. It also implies the increased need to interchange extracts from these databases.

When documents and objects of all kinds can be sent to and managed at a host as well as in dedicated outboard systems, the need to tap the central databases directly increases. Professionals who enter information into or extract information from a database also want to use their workstation's accuracy and flexibility for the electronic communication of such information.

However, the data found in databases or files is not immediately suited for free interchange between independent products, because typically it is not architected but has an implied structure and meaning. Since every data file and the related application programs can have their own convention about data representation and meaning, a common implicit architecture does not exist. The syntax and semantics of such data is only known to the programs familiar with the data, because of convention or independent communication, not because of architected control information accompanying the data.

2.2 What is FD:OCA

Formatted Data Object Content Architecture (FD:OCA) is designed to solve the above problems. FD:OCA can be viewed as a language that makes it possible to express the present format and meaning, as far as relevant, of any given data item. Format and meaning here refer to those aspects that are relevant for a program in a given environment, namely the data type and its representation. FD:OCA constructs can express such properties and can attach them to the data. The communication gap between products or environments with different data types and data representation methods is bridged using FD:OCA.

The term formatted data was coined when electronic text processing began. At that time, the major distinction between text and traditional data processing data was that text data was unformatted, while the latter had a fixed and strict format. In this document, we use the term formatted data for traditional data-processing-type data, and for any data that has an unarchitected but known format and meaning, and needs a corresponding description.

Typically, formatted data comes from, or is intended for, databases and files. Interchange of formatted data may be part of interchanging electronic mixed-object documents. But it may also occur as a process in itself — for instance when exporting or importing files, or when passing parameters from one application program to another, in the same or a different node of a network. (For the remainder of this document, the term database is used to mean small or large data collections, with or without internal structure and interdependences; in other words, simple files are always included when we talk about databases.)

2.3 A Scenario

Figure 2-1 illustrates how a user, with the aid of an editor program, would manipulate a report containing various references to variables. The variables come from a database and accompany the report as a tabular resource, along with a description, called FD:OCA Descriptor.

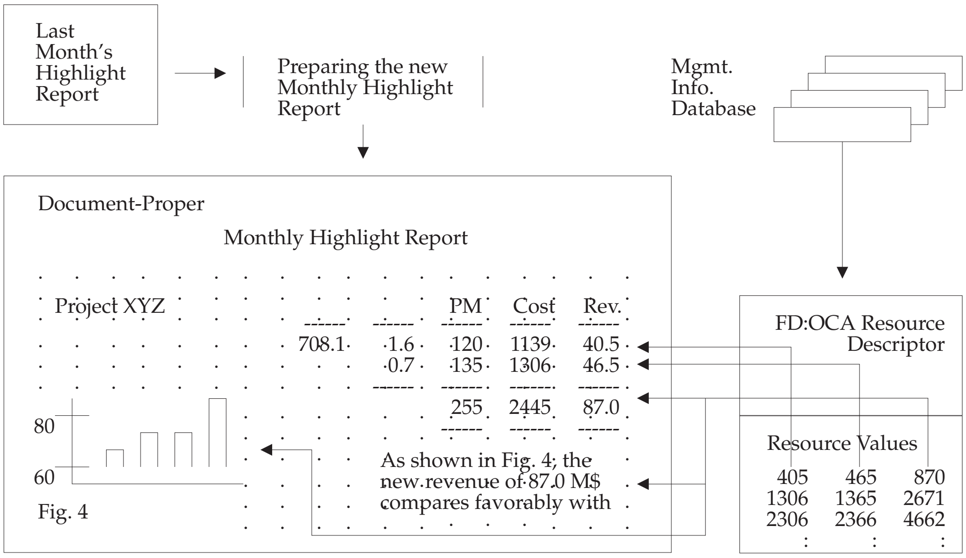

Figure 2-1 Typical FD:OCA Example

The editor program understands the references to the variables and their descriptions, and resolves the references. It performs the conversion into presentable form, if necessary, and inserts the values into their proper places. The report, with its tabular resource, could be passed to a supporting editor program in another environment, and still be processed in the same way.

If the tabular resource changes, maybe because a new extract from the database is being obtained, the report would reflect this change, showing the new values in all places where they are referenced.

The tabular resource may also change if it is obtained from a different database, perhaps with different data types and different data representations. The FD:OCA Descriptor accompanying the resource data ensures that the data values are still treated correspondingly in the report.

2.4 How to Use this Book

2.4.1 Syntax Diagrams

Throughout this volume, the syntax for FD:OCA is shown in tables using the following format:

Data (Definition of Structure)

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
(The field's offset, data type, or both)		(Name of field if applicable)	(Range of valid values if applicable)	(Meaning or purpose of the data element)	(M or O)	(Y or N)	(Syntax Exceptions)

Certain fields may be denoted as reserved. A reserved field is a parameter that has no functional definition at the current time, but may be defined at some time in the future. All bytes in a reserved field should have a value of zero. Additional columns appear to the right of the *Meaning* column. These columns are:

M/O Mandatory or optional.

DEF *Y* means that the field defaults to a value specified by FD:OCA.

N means that there is no default.

EXC The possible syntax exceptions are specified.

See Section 4.5.1.1 on page 67 for further details.

The syntax includes six basic data types:

CODE Architected constant

CHAR Character string

BITS Bit string

UBIN Unsigned binary

SBIN Signed binary

UNDF Undefined type.

Refer to Section 4.1 on page 25 for a detailed description of the syntax.

The following is an example of FD:OCA syntax for Row Layout (RLO):

Syntax

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
0	UBIN	LENGTH	6 - 255	Length of triplet	M	N	X'02'
1	CODE	TYPEID	71	Triplet type ID: Row-Layout	M	N	X'44'
2	CODE	ID	1 - 255	Construct identity	O	N	X'00'
A repeating group in the following format:							
0	CODE	LLID	1 - 255	Lower-level identifier	M	N	X'24'
1	UBIN	CNTELE	1 - 255	Count of elements taken	O	Y	X'02'
2	UBIN	REPFAC	0 - 255	Repetition factor	M	N	X'04'

2.4.2 Notation Conventions

Throughout this volume, the following notation conventions apply:

- Bytes are numbered from left to right beginning with byte 0, which is considered the high-order byte position. For example, a three-byte field would consist of byte 0, byte 1, and byte 2.
- Each byte is composed of eight bits.
- Bits in a single byte are numbered from left to right beginning with bit 0, the most significant bit, and continuing through bit 7, the least significant bit.
- When bits from multiple consecutive bytes are considered together, the first byte, byte 0, contains bits 0 to 7, and byte n contains bits $(n \times 8)$ to $((n \times 8) + 7)$.
- A negative number is expressed by the two's-complement form of its positive number. The two's-complement of a number is obtained by first inverting every bit of the number and then adding one to the inverted number.

2.4.3 Related Architecture

Character Data Representation Architecture (CDRA) defines a set of identifiers, services, supporting resources, and conventions to achieve consistent representation, processing, and interchange of graphic character data. (Graphic character data is not to be confused with the *graphic* data type, which is used to represent double-byte data in some programming languages.)

2.4.4 Industry Standards

FD:OCA supports data types defined by the following industry standards:

- *ANSI/IEEE Std. 745-1985, Binary Floating Point Arithmetic*.

Chapter 3

Overview of FD:OCA

This chapter:

- Describes the concept of FD:OCA
- Provides a description of the FD:OCA constituents and characteristics

3.1 Concepts

FD:OCA enables the description of data from databases and traditional application programs, called formatted data, so that it can be interchanged within or across environments. Such data typically has an implicit structure and meaning; FD:OCA is used to explicitly attach the information needed to understand the data.

A Formatted Data Object (FDO) consists of two components: a descriptor and a value.

- The descriptor describes the object. It expresses what format and structure the object value has; that is, what data type and representation is used for the individual parts, and how they together make up the value. This may include information on how several individual items are grouped together into arrays of data, or how sequences of fields and substructures make up records and files.
- The second part of a Formatted Data Object, the value, contains the described data. Except for the constructs defining where the value begins and ends, there are no further architectural constructs intermixed with the data. The data items occur as they have been read from the database or as they would be recorded in the database.

3.2 Constituents

This section describes the general constituents of a Formatted Data Object (FDO)—its constructs, the data types being supported, and the data structure being viewed as generalized arrays. It also defines some terms used throughout this document.

3.2.1 Constructs

Depending on the interchange purpose, the Formatted Data Objects are embedded in architected constructs of another higher-level architecture, such as the Distributed Relational Database Architecture (DRDA). The embedding architecture identifies and brackets a Formatted Data Object and its components (as appropriate) in their syntax.

The discussion here uses a generic format to suggest how the embedding architecture might convey where a Formatted Data Object and its components begin and end.

The generic format assumes that the descriptor and data (or value) components are each built from one or more Structured Fields (SF). A Structured Field is a self-identifying construct, beginning with an Introducer that delimits its scope and identifies the nature of its contents. The Introducer is followed by the contents proper. Depending on the component size, just one Structured Field will normally suffice; additional Structured Fields are used to carry segments of a component that is too big for a single Structured Field.

Other constructs, such as Begin and End indicators, may be necessary to bracket the Descriptor and Data Structured Fields.

Thus, a Formatted Data Object in this generic format may consist of these elements:

- Begin indicator
- One or more FDO-Descriptor Structured Fields
- One or more FDO-Data Structured Fields
- End indicator

Figure 3-1 on page 13 shows the structure of a typical Formatted Data Object. FD:OCA is concerned with the contents of the Descriptor and Data Structured Fields. The Structured Field Introducers or any bracketing constructs may be different in different surrounding architectures; they are not addressed by FD:OCA.

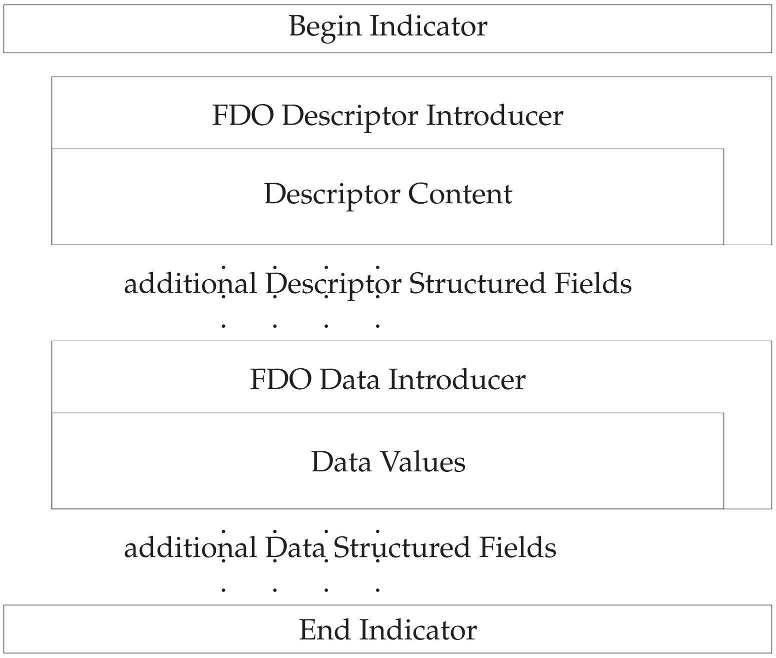

Figure 3-1 Formatted Data Object

The Descriptor component contains the FD:OCA constructs describing the Data component. These constructs are called Attributes or Attribute Triplets. Like Structured Fields, they are self-identifying constructs. Each has a length field and a type field forming the Introducer for the last part, the attribute proper. The term triplet refers to the fact that these constructs consist of three pieces: a length field, a type field, and parameter data.

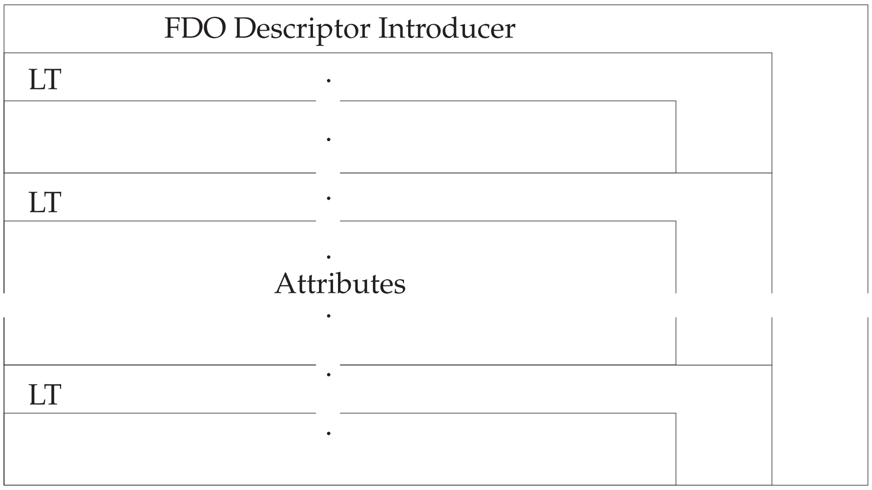

Figure 3-2 FDO Descriptor

Figure 3-2 shows how the Attribute constructs are carried in a Descriptor Structured Field. Each Attribute has an Introducer, labeled LT, for Length and Type. One or more Attributes be needed. The attribute contents together carry the information necessary to describe the format and meaning of the Data component.

3.2.2 Data Types

FD:OCA supports the following data types and representation methods:

1. Bit and byte strings; that is, strings without additional semantics, in fixed and variable-length forms.
2. Character strings in fixed and variable-length forms, with Coded Character Set parameters supplying the semantics.
3. Numeric data that can describe integer numbers and then appear in various representations and lengths as either:

   — Decimal numbers in zoned or packed format

   — Signed or unsigned binary numbers

   Numeric data that can describe numbers with fractional parts in the notations:

   - Decimal or binary fixed point
   - System/390 hexadecimal floating point
   - IEEE binary floating point
   - VAX binary floating point

   This numeric data can also appear in various representations and lengths.

3.2.3 Data Arrays

The data arrangements described through FD:OCA cover a broad spectrum. They include single fields and numbers as well as tabular arrays and collections of records from traditional databases. These seemingly different arrangements are treated by FD:OCA as variations of a single general concept, as explained below. Conceptually, each FD:OCA object is a multi-dimensional, more or less regular array of individual elements, with individual formats that may or may not be identical. As an illustration, consider a file with several records of different lengths and different field layouts, as shown in Figure 3-3. Such a situation could be treated in FD:OCA as a two-dimensional irregular array. The FD:OCA Descriptor for this structure would have individual descriptions for most of the individual fields, and would then describe each record layout, one after the other, referencing the field descriptions. The last three records, having an identical layout, would share a description.

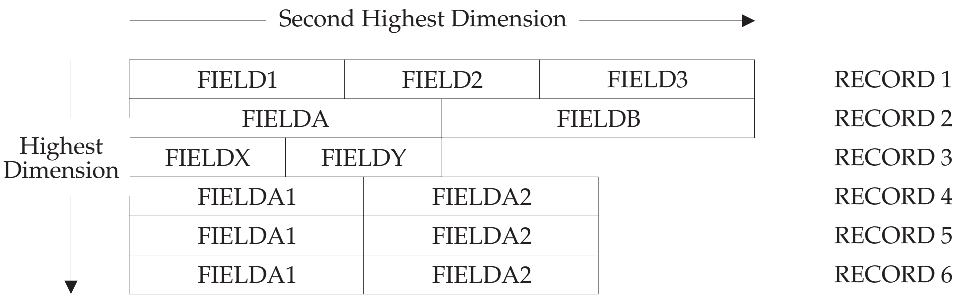

Figure 3-3 General Array Example

Variations of this concept with less diversity and overhead would be used to describe more regular data arrangements, such as files with records of fixed-length and uniform layout, or rectangular matrices with fields of identical format.

Those objects that are actually single fields with no array-like substructure would be treated as trivial arrays with just one element.

The next section of this chapter introduces and defines some specific terms related to arrays; the subsequent sections then use examples to show how this concept of describing a general array is applied to specific cases.

3.2.4 Definition of Terms

FD:OCA is concerned with describing certain interchange objects carried in a sequential, linear data stream, and hence with objects that are themselves just linear sequences of bits carrying encoded information. The contents of an object are not physically arranged in some rectangular fashion. Therefore, discussing dimensions of arrays in this context requires some imagination and deserves a definition.

3.2.4.1 Partitions, Dimensions, and Extents

As illustrated in Figure 3-4, a linear string of fields may be thought of as being partitioned in a hierarchical way. A first level of partitioning divides the string into a certain number of partitions. On the next level, some or all of these partitions are further divided into sub-partitions, which in turn may be further divided into sub-sub-partitions on the third level, and so on.

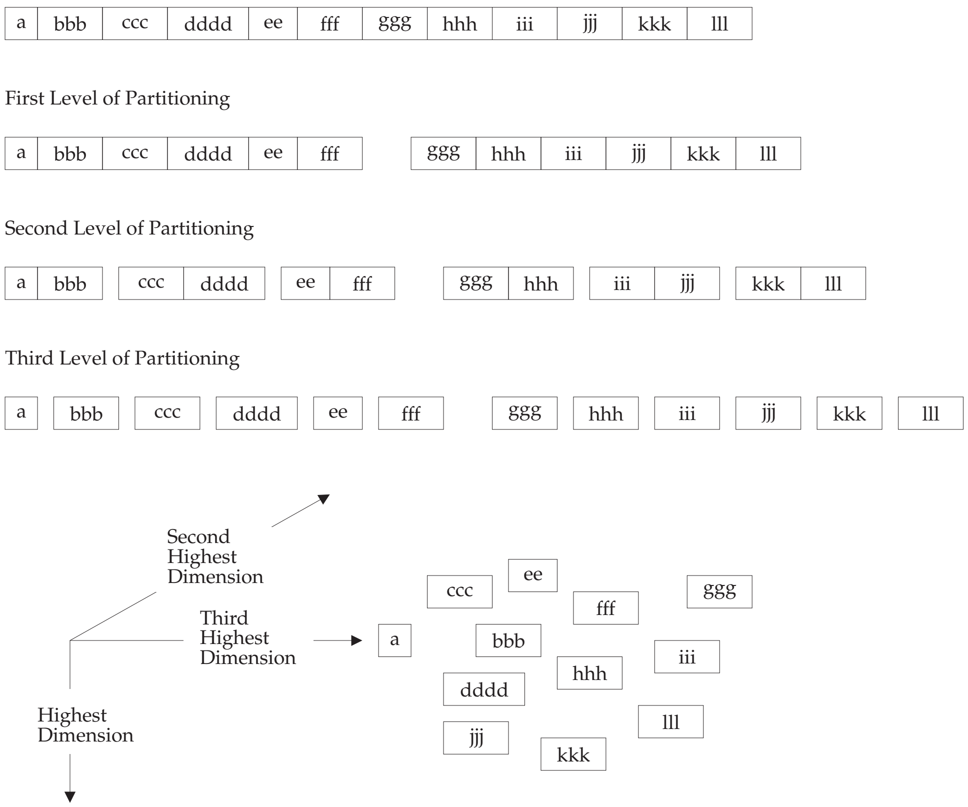

Figure 3-4 Partitioning a Linear String of Fields into Three Dimensions

Each of these levels of partitioning is called a dimension. The first level of partitioning is called the highest dimension, the second is called the next-highest or second-highest dimension, the last one is called the lowest dimension. If n levels of partitioning are defined, the string is sometimes called an n-dimensional array.

As suggested in the figure, a string of fields partitioned this way can be thought of as being arranged in space in a three-dimensional style, hence the term dimension.

In this example, the partitions of any particular level are treated alike; they are divided into the same number of sub-partitions: namely two on the first level, three on the second, and again two on the third level. If all partitions of a particular level are divided into the same number of sub-partitions, then this number is called the extent of the dimension. The extents of the three dimensions in the example are 2, 3, and 2, respectively.

If partitions on one level are subdivided in different ways, the number of sub-partitions of any given partition is called a local extent. A dimension then has an extent of k, if all its local extents are equal to k.

Arrays in which all dimensions have well-defined extents are called regular arrays.

In this book, the dimensions of regular arrays are sometimes referred to by sequence numbers. For example, the highest dimension of a three-dimensional regular array is called Dimension 3; the next highest is Dimension 2, and Dimension 1 is the lowest dimension.

3.2.4.2 *Subarrays*

For regular arrays, all dimensions have well-defined extents. Each individual item of a regular n-dimensional array is characterized by n location numbers, expressing where that item is located relative to each of the n dimensions.

In a regular two-dimensional array, the highest-level partitions are also called rows. Each one is a subarray consisting of all those items that have a particular location number in the high dimension.

A subarray of a regular two-dimensional array that consists of all items located at a particular position in the low dimension is called a column.

For a regular three-dimensional array, a subarray consisting of all those items located at a particular position in one of the dimensions is called a plane.

For a regular n-dimensional array, a subarray consisting of all those items located at a particular position in one of the dimensions is called a slice. Planes are special cases of slices of three-dimensional arrays, as rows or columns are the slices of two-dimensional arrays.

The dimensionality of an array, the number of dimensions it has, depends on the viewpoint. For certain discussions it may be useful to disregard some dimensions. Thus an n-dimensional array may also be viewed as a one-dimensional array or vector of certain entities, known to have a (n–1) dimension structure, or as a two-dimensional arrangement of certain (n–2)-dimensional arrays, and so on.

An array may even be viewed simultaneously in several logical directions. For each of the dimensions of a regular array, the array may be viewed as a sequence of slices, each of which is characterized as being located in a particular position in that dimension. If the dimension has an extent of k, then the array can be viewed as a sequence of k slices defined by that dimension. If another dimension has an extent of m, then the array can also be seen as a sequence of m slices defined by and in the direction of that dimension.

3.3 Characteristics

This section illustrates the characteristics of the architecture by means of some simple examples of Formatted Data Objects.

3.3.1 Describing Data Arrays and Data Types

Figure 3-1 on page 13 and the subsequent discussions indicated how the FD:OCA Descriptor describes an FD:OCA object. The Descriptor contains self-identifying constructs called attribute triplets. The attribute triplets describe the structure and properties of the Formatted Data Object values. Several triplets may occur.

Using short labels, called local identifiers (LIDs), triplets may refer to other triplets, which in turn may refer to yet further triplets, and so on.

The examples that follow show how the attribute triplets are used to express information on data types and structure.

3.3.2 Examples

This section illustrates the typical usage of the attribute triplets. These examples demonstrate the essential purpose of the architecture constructs.

For readability, the examples ignore details like code points and length field values and rather try to convey those through symbolic descriptions and graphical methods. For instance, logically existing boundaries between fields of a record are suggested through blank spaces, which would typically not exist in a real database record. Also, numbers in the examples are always printable and therefore readable. Thus, the examples should not be taken too literally. They are meant to convey the FD:OCA concepts.

In addition to the graphical representation, the examples are also shown using an abstract syntax notation, offering a more program-oriented point of view.

Among the attribute triplets to be discussed are:

Simple Data Array
: Used to describe Formatted Data Object values that are either single items or regular arrays of several such items, each having the same type and format. Type and format are also described in this construct.

Row Layout
: Used to describe irregular arrays, in which the elements are not all of the same data type. Typically, several of these triplets are needed. They may refer to each other and to Simple Data Array triplets.

Group Data Array
: An alternative way of describing fields and structures, with override options for data type information.

Metadata Definition
: A means to tag data descriptions with information specific to certain classes of applications.

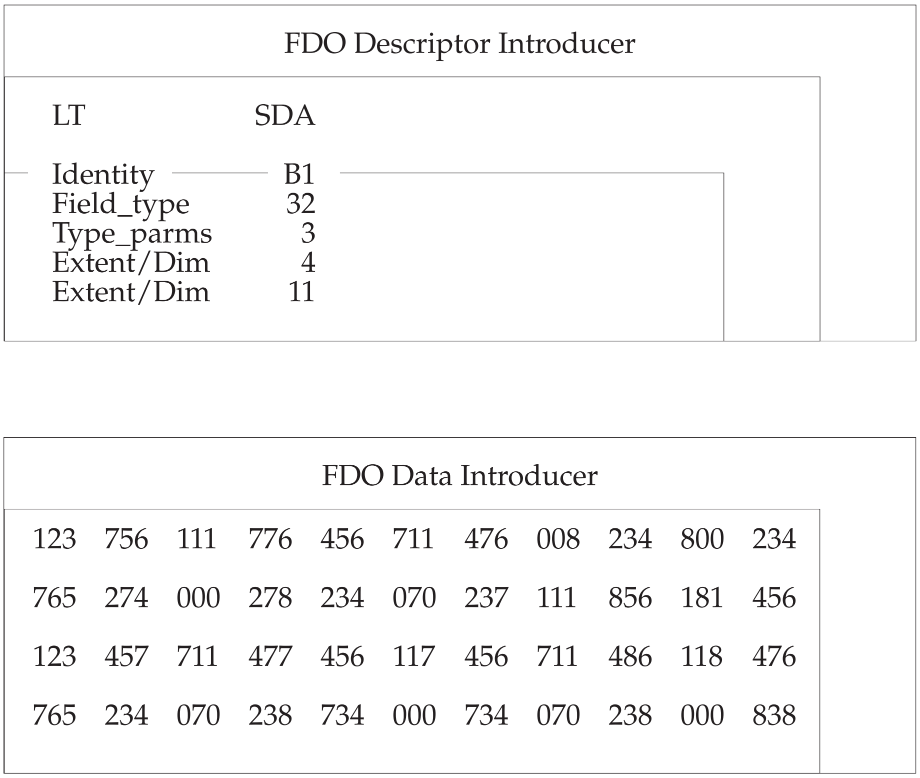

Figure 3-5 Regular Array of Three-Digit Numeric Fields

```
Begin FDO 'OBJECTA'

      FDO Descriptor

          SDA  ID = X'B1'
               Field_type = X'32', Type_parms = 3, Extent/Dim = 4,
               Extent/Dim = 11

      FDO Data

          123 756 111 776 456 711 476 008 234 800 234
          765 274 000 278 234 070 237 111 856 181 456
          123 457 711 477 456 117 456 711 486 118 476
          765 234 070 238 734 000 734 070 238 000 838

End FDO
```

Figure 3-5 shows a 4-by-11 array of three-digit numbers. The Simple Data Array (SDA) triplet is the only triplet here. It says that the data type is some kind of numeric, and that there are several dimensions. A first Extent/Dim (Extent-per-Dimension) entry describes the extent of the highest dimension as being 4. The next entry says that the second dimension has an extent of 11. And an entry called Type Parameters expresses that the numbers have a length of 3 digits. This is a particularly short descriptor, needing only one attribute triplet, due to the fact that the data is very regular.

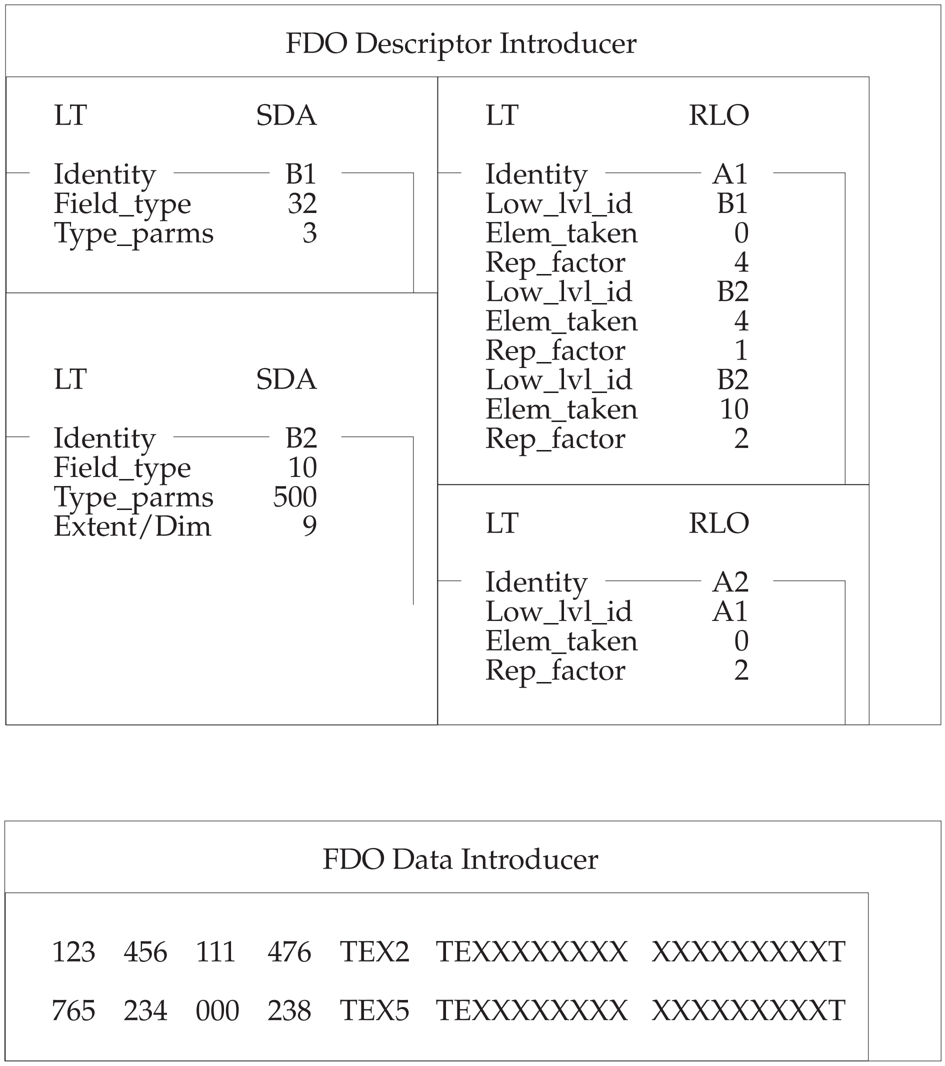

Figure 3-6 Regular Array, Several Field Types

```
Begin FDO 'OBJECTB'

      FDO Descriptor

          SDA  ID = X'B1'
               Field_type = X'32', Type_parms = 3

          SDA  ID = X'B2'
               Field_type = X'10', Type_parms = 500,
               Extent/Dim = 9

          RLO  ID = X'A1'
               Low_lvl_id = X'B1', Elem_taken =  0, Rep_factor = 4,
               Low_lvl_id = X'B2', Elem_taken =  4, Rep_factor = 1,
               Low_lvl_id = X'B2', Elem_taken = 10, Rep_factor = 2
```

```
        RLO  ID = X'A2'
             Low_lvl_id = X'A1', Elem_taken =  0, Rep_factor = 2

    FDO Data

        123 456 111 476  TEX2 TEXXXXXXXX XXXXXXXXXT
        765 234 000 238  TEX5 TEXXXXXXXX XXXXXXXXXT

End FDO
```

Figure 3-6 on page 19 also has a regular array, but with fields of different data types and different lengths. This example needs two Row Layout (RLO) triplets and two SDAs. One of the RLOs, labeled X'A1', is a layout description for the rows of this two-row table. Through a sequence of descriptive entries, it describes the fields making up a row.

A first group of entries expresses that the row starts with a numeric field, described through SDA X'B1', and that this field type is repeated 4 times. A parameter called *Elem_taken* occurs, but is not used in this group.

A next group of three entries refers to X'B2', describing a character string. Here, the Elem_taken entry is used. It is an overriding length specification, and says that the length needed here is 4, regardless of what X'B2' specifies. The repeating factor 1 says that this field occurs just once.

The last group of entries refers to the same X'B2' and expresses that two fields of this type occur.

While the SDA labeled X'B1' could describe a whole simple data array (as its name suggests), it serves here as just a single numeric field description. SDA X'B2', on the other hand, describes a slightly more complex structure: a one-dimensional array or string, made up of 9 characters. As with X'B2' above this specification may be overridden when referring to this description.

The other RLO, X'A2', refers to X'A1' and says that the description X'A1' is also used for this row. Essentially, X'A2' defines how the array is arranged in its highest dimension.

FDO Descriptor Introducer

LT	SDA
Identity	B1
Field_type	32
Type_parms	4
Extent/Dim	7

LT	RLO
Identity	A1
Low_lvl_id	B1
Elem_taken	8
Rep_factor	1
Low_lvl_id	B1
Elem_taken	5
Rep_factor	1
Low_lvl_id	B1
Elem_taken	7
Rep_factor	3

FDO Data Introducer

7745	1229	1947	2345	1235	5681	3947	1234
2371	1257	1278	5681	1257			
2375	1237	2947	2345	1537	5681	4947	
6814	1237	1247	1234	4237	5481	5947	
5437	5681	2345	1237	5681	1947	6234	

Figure 3-7 Irregular Array, All Numeric Fields

```
Begin FDO 'OBJECTC'

      FDO Descriptor

          SDA  ID = X'B1'
               Field_type = X'32', Type_parms = 4, Extent/Dim. = 7

          RLO  ID = X'A1'
               Low_lvl_id = X'B1', Elem_taken = 8, Rep_factor = 1,
               Low_lvl_id = X'B1', Elem_taken = 5, Rep_factor = 1,
               Low_lvl_id = X'B1', Elem_taken = 7, Rep_factor = 3

      FDO Data

          7745 1229 1947 2345 1235 5681 3947 1234
          2371 1257 1278 5681 1257
          2375 1237 2947 2345 1537 5681 4947
          6814 1237 1247 1234 4237 5481 5947
```

```
            5437 5681 2345 1237 5681 1947 6234

    End FDO
```

In Figure 3-7 on page 21 we have an irregular array, but with only one data type. The data type, described through X'B1', is a four-digit numeric type. Actually, it is a one-dimensional array of seven four-digit numbers. The only RLO, labeled X'A1', describes how a two-dimensional array is arranged in its highest dimension by stacking one-dimensional arrays. The first group of entries refers to X'B1', and gives an overriding Elem_taken value of 8. This number overrides the 7 shown in X'B1' and describes a row of 8 four-digit numbers. The repetition factor says this type of row occurs just once. The next group of entries also refers to X'B1', this time with an overriding count of 5, since the second row has only five numbers. The last group refers to X'B1' with an overriding value of 7, which in this case means 0, because 7 is the number shown as highest extent in X'B1'. It then has a repetition factor of 3, so that it describes three rows.

Thus, this example describes an irregular array of five rows, all of which consist of some number of four-digit numbers. The first row has 8, the second row has 5, and the last three rows have 7 numbers each. Two attribute triplets were needed here.

FDO Descriptor Introducer

LT	SDA
Identity	B1
Field_type	10
Type_parms	500
Extent/Dim	3

LT	RLO
Identity	A1
Low_lvl_id	B1
Elem_taken	1
Rep_factor	1
Low_lvl_id	B1
Elem_taken	0
Rep_factor	1

LT	RLO
Identity	A2
Low_lvl_id	B1
Elem_taken	0
Rep_factor	1
Low_lvl_id	B1
Elem_taken	4
Rep_factor	1

LT	RLO
Identity	A3
Low_lvl_id	B1
Elem_taken	2
Rep_factor	1
Low_lvl_id	B1
Elem_taken	0
Rep_factor	1

LT	RLO
Identity	A4
Low_lvl_id	B1
Elem_taken	0
Rep_factor	2

LT	RLO
Identity	A5
Low_lvl_id	A1
Elem_taken	0
Rep_factor	1
Low_lvl_id	A2
Elem_taken	0
Rep_factor	1
Low_lvl_id	A3
Elem_taken	0
Rep_factor	1

LT	RLO
Identity	A6
Low_lvl_id	A4
Elem_taken	0
Rep_factor	3

LT	RLO
Identity	A7
Low_lvl_id	A5
Elem_taken	0
Rep_factor	1
Low_lvl_id	A6
Elem_taken	0
Rep_factor	1

FDO Data Introducer

a	bbb	ccc	dddd	ee	fff
ggg	hhh	iii	jjj	kkk	lll

Figure 3-8 Three-Dimensional Array

```
Begin FDO 'OBJECTE'

      FDO Descriptor

          SDA  ID = X'B1'
               Field_type = X'10', Type_parms = 500,
               Extent/Dim = 3

          RLO  ID = X'A1'
               Low_lvl_id = X'B1', Elem_taken = 1, Rep_factor = 1,
               Low_lvl_id = X'B1', Elem_taken = 0, Rep_factor = 1

          RLO  ID = X'A2'
               Low_lvl_id = X'B1', Elem_taken = 0, Rep_factor = 1,
               Low_lvl_id = X'B1', Elem_taken = 4, Rep_factor = 1

          RLO  ID = X'A3'
               Low_lvl_id = X'B1', Elem_taken = 2, Rep_factor = 1,
               Low_lvl_id = X'B1', Elem_taken = 0, Rep_factor = 1

          RLO  ID = X'A4'
               Low_lvl_id = X'B1', Elem_taken = 0, Rep_factor = 2

          RLO  ID = X'A5'
               Low_lvl_id = X'A1', Elem_taken = 0, Rep_factor = 1,
               Low_lvl_id = X'A2'= 0, Rep_factor = 1,
               Low_lvl_id = X'A3', Elem_taken = 0, Rep_factor = 1

          RLO  ID = X'A6'
               Low_lvl_id = X'A4', Elem_taken = 0, Rep_factor = 3

          RLO  ID = X'A7'
               Low_lvl_id = X'A5', Elem_taken = 0, Rep_factor = 1,
               Low_lvl_id = X'A6', Elem_taken = 0, Rep_factor = 1

      FDO Data

          a   bbb    ccc dddd      ee  fff
          ggg hhh    iii  jjj      kkk lll

End FDO
```

Figure 3-8 on page 23 describes the three-dimensional array discussed earlier with Figure 3-4 on page 15. The SDA triplet labeled X'B1' describes a string of 3 characters as a one-dimensional array having an extent of 3. The triplets labeled X'A1', X'A2', and X'A3', refer to X'B1'. By using the overriding *Elem_taken* parameter, they describe individual rows of two character strings each, with different string lengths. They describe the rows that make up the upper plane in Figure 3-4 on page 15. Triplet X'A4' refers to X'B1' without overriding its extent but with a repetition factor of 2. It thus describes a row of two character strings of length 3 each. The RLO triplet labeled X'A5' refers to X'A1', X'A2', and X'A3', and provides a description of the upper plane in Figure 3-4 on page 15. The lower plane is represented by X'A6', calling for three rows of style X'A4'. The X'A7' triplet puts the two planes together.

Chapter 4

FD:OCA Specifications

This chapter:

- Introduces the conventions used in FD:OCA
- Describes the constituents of an FD:OCA object
- Describes the syntax and semantics of the FD:OCA constructs
- Provides definitions for exception conditions

4.1 Conventions Used in FD:OCA Specifications

This section describes the syntax conventions used in the FD:OCA specifications.

The syntax of each FD:OCA construct is described with the aid of a table, as illustrated in Table 4-1. The semantics associated with the parameters and parameter values appear below the figure.

Syntax

Table 4-1 Syntax Description of Structured Fields and Triplets

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
A repeating group in the following format:							

In the table:

- Offset refers to the position, indexed on zero, of a parameter within the construct.
- Type denotes the syntax of the parameter. Types are:
  - — CODE refers to a parameter for which each valid value has a distinct meaning.
  - — CHAR means that the parameter provides a name.
  - — BITS means bit string and refers to a parameter composed of collections of small numbers, usually one, of consecutive bits; each collection of consecutive bits is interpreted as a code, in the sense described above for CODE.
  - — UBIN refers to a numeric parameter that can be interpreted arithmetically. It is a one or two byte unsigned binary number.
  - — SBIN refers to a numeric parameter that can be interpreted arithmetically. It is a one or two byte signed binary number.
  - — UNDF means undefined and refers to a parameter for which there is no syntactic or semantic definition, or to a parameter string composed of several parameters with the syntax for each parameter specified below the table.

- Name is the name used in this architecture specification to refer to the parameter.
- Range specifies the valid range of values for a parameter. If no value is specified in the syntax table, the valid entries are listed in the parameter description.
- Meaning gives a short description of the parameter. A reserved field is a parameter that has no functional definition at the current time. Its value must be zero.
- M/O refers to whether this parameter in the structured field must be specified:

  — O means that the parameter specification is optional.

  — M means that the parameter specification is mandatory.

  When a positional parameter is optional and its value range does not include a value of all-zero bits, then a value of all-zero bits is permissible and is interpreted as *parameter not specified*. In this case the default value is used. If one or more positional parameters at the end of a construct are optional, then they may also be left off.
- DEF refers to the existence of an architecture-defined default for the parameter:

  — N means that there is no default value.

  — Y means that there is a default value and it is given below the table.
- EXC indicates what syntax exceptions are to be expected for any particular parameter. The value shown is a two-digit hexadecimal number, to be read as eight bit-flags. The eight bits from left to right correspond to general exception conditions with exception identifiers 1 through 8; for example, an 07 means that exceptions with identifiers 6, 7, and 8 may be detected, but not those with identifiers 1, 2, 3, 4, and 5. See Section 4.5.1.1 on page 67 for a definition of the exception categories and their identifiers.

A repeating group of one or more parameters can be specified at the end of a construct. Descriptive material indicates what, if any, restrictions apply to the number of times the repeating group can appear in the construct.

Except for the last, each occurrence of a repeating group must consist of all parameter specifications belonging to this group, independent of whether the parameter is optional or mandatory. Only in the last repeating group, trailing optional parameters may be left off.

The Offset specification of a repeating group is reset to 0.

Unless stated otherwise, items in a sequence of bits or bytes are addressed starting with 0 for the leftmost or low-addressed bit or byte. Item 1 is adjacent to and to the right of item 0, and so on. The leftmost bit in a byte or sequence of bytes is sometimes also called the high-order bit. The rightmost bit is also called the low-order bit.

4.2 FD:OCA Object Constituents

Each FD:OCA object has a Descriptor and an optional Data part. Depending on the interchange purpose, the Formatted Data Objects are embedded in the architected constructs of another higher-level architecture, such as the Distributed Relational Database Architecture (DRDA). The embedding architecture identifies and brackets a Formatted Data Object and its components, as appropriate in its syntax.

The discussion here uses a generic format to suggest how the embedding architecture might convey where a Formatted Data Object and its components begin and end.

The generic format assumes that both the Descriptor and the Data component are each built from one or more Structured Fields (SF). A Structured Field is a self-identifying construct, beginning with an Introducer, that delimits its scope and identifies the nature of its contents. The Introducer is followed by the contents proper. Depending on the component size, just one Structured Field will normally suffice; additional Structured Fields serve to carry segments of a component that is too big for a single Structured Field. The Descriptor consists of one or more Descriptor Structured Fields, and the Data part consists of zero or more Data Structured Fields.

The Descriptor Structured Fields describe the structure and appearance of the object through the attribute triplets.

The actual data of the object appears in the Data Structured Fields. The Data Structured Field content is pure data without any additional, architecturally prescribed, constructs.

The complete sequence of Descriptor Structured Fields of an object is called the Descriptor. The sequence of all Data Structured Fields is called the Data part or the value of that object.

4.3 FD:OCA Descriptor Component Content

The FD:OCA Descriptor contains the information that defines a Formatted Data Object (FDO) as a sequence of attribute triplets. The attribute triplets describe the structure and properties of the FDO values.

Generally, an attribute triplet consists of three parts: a one byte length field, a one-byte type field, and up to 253 bytes of parameter data.

Syntax

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
0	UBIN	LENGTH	2 - 255	Length of triplet	M	N	X'02'
1	CODE	TYPEID		Triplet type ID	M	N	X'44'
2 - n				Parameter data			

Semantics

Length of triplet specifies the length in bytes of the triplet, including this one byte length field.

Triplet Type ID is a one byte code identifying the triplet type.

Parameter data consists of one or more parameters. The number of parameters, the length, and the structure of each parameter in the triplet is dependent on the triplet type.

4.3.1 Descriptor Attribute Triplets

Several different kinds of attribute triplets are available to describe the structure and properties of a Formatted Data Object. In a simple case, the description may consist of just a single attribute triplet; more typically, though, several attribute triplets together form the description. In such cases, a major triplet refers to lower-level triplets, which in turn may refer to yet lower-level triplets, and so on. The major attribute triplet is identifiable by the fact that it is not referenced by any other triplets. Only one attribute description may exist; in other words, no more than one major or unreferenced attribute triplet may occur.

4.3.1.1 References

To establish a connection between them, attribute triplets refer to other attribute triplets inside the same FD:OCA object through the concept of a Local Identifier (LID). For this purpose, the attribute triplets carry a one byte field called Identity. This makes it possible to refer from one triplet to another by specifying the identity of the intended triplet. Thus, from within an attribute triplet, other attribute triplets in the same object can be referred to by a one-byte label called LID.

All LID references must follow this position rule: A referenced LID must have been defined in the object to the left of (or prior to) the reference. If more than one triplet in the Descriptor contains the referenced LID, then the reference resolves to the one occurring nearest to and prior to the reference.

The LID assigned to the referencing triplet as well as all LID definitions succeeding this triplet in the object are not in the scope of this LID reference.

4.3.1.2 *Simple Data Array (SDA)*

A Simple Data Array triplet is used to describe those parts of FD:OCA object values that are either single items or linear or rectangular arrays of several such items, each having the same format.

Syntax

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
0	UBIN	LENGTH	4 - 254	Length of triplet	M	N	X'02'
1	CODE	TYPEID	X'70'	Triplet type ID: Simple Data Array	M	N	X'44'
2	CODE	ID	1 - 255	Construct identity	O	N	X'00'
3	CODE	FTYPE		Field type	M	N	X'06'
4 - 11	UNDF	TPARM		Type parameters	O	Y	X'02'
A repeating group in the following format:							
0 - 1	UBIN	EXTENT	0 - 32767	Extent per dimension	O	N	X'44'

Semantics

This triplet is a declarative for the attributes of parts of an FD:OCA object value with uniform structure. It is used for single items, such as single numbers, or for values that are vectors or rectangular, or more-dimensional arrays of single items of identical format, all having the same field length, field type, and type parameter (for example, being binary numbers of a certain length).

ID provides for a local name which allows this triplet to be referenced from other triplets.

FTYPE describes the data type of one field. Valid entries for this parameter depend on the supported FD:OCA subset and can be found under the SDA description for this specific subset in Section 5.2 on page 73. See Section 4.3.3 on page 36 for detailed data type descriptions.

TPARM provides additional information regarding the field type. Valid contents depend on the field type. See Section 4.3.3 on page 36 for more details. A typical parameter is the field length of the single items making up the array described. The default is as specified in the data type definition.

The repeating group may occur zero, one, or more times. It contains the following parameter:

EXTENT: This repeatable entry is used, if necessary, to describe the structure and size of arrays of data. Each entry defines an array dimension and expresses how many addressable entities exist in that dimension. Indexed access to elements or parts of data arrays is defined with reference to these dimension specifications.

A series of Extent-per-Dimension entries can be viewed as partitioning the described FD:OCA object value, or part thereof, into hierarchically nested partitions. The first entry expresses into how many pieces of equal length the whole described entity is to be partitioned; the next entry, if present, says for each of those pieces into how many sub-pieces of equal length it is partitioned, and so on.

The last Extent-per-Dimension entry expresses how many items of described data type and length comprise the lowest hierarchical partition. If no Extent-per-Dimension entry is specified, then a single item is being described.

A parameter value of zero means that the extent for this dimension is not explicitly specified and must therefore be derived from the values. A zero can only be specified for the first occurrence of this parameter in a triplet which is not referenced from another attribute triplet. Only the extent of the highest dimension may use implicit specification through the values; otherwise an exception condition with exception-id 10 occurs. For all invalid values of zero for this parameter, a value of 1 is assumed.

The implicit specification of the extent cannot be used with data types that do not consume any value space, such as a fixed-length character string of zero length. In this case, an exception condition with exception-id 10 occurs.

4.3.1.3 *Row Layout (RLO)*

The Row Layout triplet is used to describe a row containing fields of different types, or to describe a table containing rows of different types, or to describe multi-dimensional entities of this nature.

Syntax

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
0	UBIN	LENGTH	6 - 255	Length of triplet	M	N	X'02'
1	CODE	TYPEID	X'71'	Triplet type ID: Row-Layout	M	N	X'44'
2	CODE	ID	1 - 255	Construct identity	O	N	X'00'
A repeating group in the following format:							
0	CODE	LLID	1 - 255	Lower level identifier	M	N	X'24'
1	UBIN	CNTELE	1 - 255	Count of elements taken	O	Y	X'02'
2	UBIN	REPFAC	0 - 255	Repetition factor	M	N	X'04'

Semantics

In reference to Figure 3-4 on page 15, a Row Layout Triplet defines how an FD:OCA object or a partition thereof is in itself thought to be partitioned, by describing how many pieces of what data type it consists of. Typically, it is only used when those pieces or sub-partitions are of different types, since otherwise an SDA construct can be used instead. It implicitly defines a local extent.

ID provides a local name, allowing this triplet to be referenced from other attribute triplets.

The repeating group may occur one or more times. It contains the following parameters.

LLID specifies the local name of another lower-level Row Layout construct, or of a Simple Data Array construct, or of a Group Data Array construct. Note that the description of complex data structures is done through nesting appropriate Row Layout, Simple Data Array, and Group Data Array constructs. An exception condition with exception-id 09 exists if the referenced triplet is not an SDA or RLO or GDA. In such a case, the value is determined as described for the general exception condition category with exception-id 3; see Section 4.5.1 on page 67.

CNTELE is an overriding local extent for the next lower level. It defines how long the lower-level row, or table, or sub-array in higher dimensions, should actually be, measured in number of partitions of the lower level. If the lower level has fewer partitions than requested here, then the last partition specification is treated as if it were repeated enough times to fill this specification. If it has more, then the excessive specifications are ignored.

This parameter is designed to minimize the number of different lower-level constructs required to describe variable-length records.

The default is the number of partitions of the lower level. If the lower level is an SDA, as opposed to RLO and GDA, then the number of partitions overridden by this parameter is the extent of its highest dimension. If no highest dimension exists, because the SDA describes a single zero-dimensional field, then this parameter is ignored.

If the referenced lower-level structure is a group described by a Group Data Array construct, then the parameter specification applies to each element of the described group, defining an overriding local extent for each of them. For zero-dimensional SDAs in the GDA, this parameter is ignored.

REPFAC specifies how often the above field, row, table, or group definition is to be repeated through repetition of the repeating group, before the description of the next field, row, table, or group begins.

A parameter value of zero means that the number of partitions is not explicitly specified and must therefore be derived from the values. A zero can only be specified for the last occurrence of this parameter in a triplet which is not referenced from another attribute triplet; only the last group of partitions in the highest dimension may use implicit specification for its number of occurrences through the values; otherwise, an exception condition with exception-id 10 occurs. For all invalid values of zero for this parameter, a value of 1 is assumed.

The implicit specification of the number of partitions cannot be used with data types that do not consume any value space, such as a fixed-length character string of zero length. In this case, an exception condition with exception-id 10 occurs.

The total number of partitions of the entity described by this RLO triplet depends on what Repetition Factors are specified in each repeating parameter group, and on how many repeating groups occur, and on what they refer to. The total number of partitions in the described entity, the local extent of the entity, is the weighted sum of the Repetition Factors in its repeating groups. The weight for a Repetition Factor is 1, if the referenced entity is an RLO or SDA. If the referenced entity is a GDA, then the weight is the group size (that is, the number of elements making up the group). In other words, the Repetition Factors for entities other than GDAs are simply added, and those specified for GDAs are first multiplied with the size of the group, and then added.

4.3.1.4 Group Data Array (GDA) and Nullable Group Data Array

The Group Data Array triplet allows the definition of a group, which is a sequence of data field or array descriptions, optionally with modified data type specifications.

Syntax

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
0	UBIN	LENGTH	4 - 255	Length of triplet	M	N	X'02'
1	CODE	TYPEID	X'75' or X'76'	Triplet type ID: Group Data Array	M	N	X'44'
2	CODE	ID	1 - 255	Construct identity	O	N	X'00'
A repeating group in the following format:							
0	CODE	REFID	1 - 255	Array reference identity	M	N	X'26'
1 - 2	UNDF	TPARM		Type parameter specification	O	N	X'00'

Semantics

This triplet constitutes a sequence of fields or arrays that can be referred to as a unit called group. In contrast to the Row Layout triplet (RLO), the described group does not form a higher-dimension entity. It remains a sequence of separate fields or arrays in the order referenced here.

If a Simple Data Array triplet is referenced from this triplet, then bytes 6 and 7 of the Type Parameters are replaced with the specified parameter value. For a description of Type Parameters, see Section 4.3.3 on page 36.

TYPE is either X'75' or X'76'; the two variants are identical, except that type X'76' additionally introduces a null indicator for the whole group. In this case, as with nullable data types for fields, a null indicator byte precedes the group, indicating in its high-order bit whether the group is present or missing. B'0' indicates presence; B'1' indicates absence.

ID provides a local name, allowing this triplet to be referenced from other attribute triplets. The effect of referencing this triplet is the same, with the possible exception of the type parameter overriding, as when referencing individually all the constructs referenced here.

The repeating group may occur one or more times. It contains the following parameters.

REFID specifies the local name of a Row Layout construct, of a Simple Data Array construct, or of another Group Data Array construct. The referenced construct becomes part of the described structure in the position determined by the occurrence of the repeating group.

An exception condition with exception-id 09 exists if the referenced triplet is not an SDA, RLO, or GDA. In such a case, the value is determined as described for the general exception condition category with exception-id 3; see Section 4.5.1 on page 67.

TPARM is an overriding specification for bytes 10 - 11 of the referenced Simple Data Array (SDA) triplet. It replaces the seventh and eighth byte of the Type Parameters in the referenced SDA, which typically is a field length. See Section 4.3.3 on page 36 for what contents are valid depending on the field type of the SDA. A parameter value of zero does not override any bytes.

This parameter is ignored if the referenced triplet is not an SDA.

4.3.1.5 *Metadata Definition (MDD)*

This triplet can precede SDA, GDA, RLO, or other MDD triplets, in order to provide the data structures with additional, application-specific attribute information through the use of metadata. Metadata consists of a metadata type, and optionally a metadata value. It can be attached to a data structure, like a tag, by placing it prior to the triplet describing the data structure. One or more MDD triplets may tag an attribute triplet. If a triplet to be tagged already has tags, then the additional tag is placed prior to the existing tags.

Table 4-2 Syntax

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
0	UBIN	LENGTH	5 - 252	Length of triplet	M	N	X'02'
1	CODE	TYPEID	X'78'	Triplet type ID: Metadata Definition	M	N	X'44'
2	CODE	ID	X'00'	Construct identity	O	N	X'00'
3	CODE	CLASS		Application class	M	N	X'02'
4	CODE	SUBTYP		Meta Data type	M	N	X'02'
5	CODE	REFTYP		Meta Data reference type	O	Y	X'22'
6	UNDF	REFID		Meta Data reference	O	N	X'20'
A repeating group in the following format:							
0	UBIN	CRITDIM	1 - 255	Criteria dimension	O	Y	X'02'
1 - 2	UBIN	LOWLIM	0 - 32767	Low index limit	O	Y	X'02'
3 - 4	UBIN	HIGHLIM	0 - 32767	High index limit	O	Y	X'26'

Semantics

ID is an unused parameter; it may contain a value of all-zero bits. The field exists only for the sake of syntactical uniformity with related constructs.

CLASS and SUBTYP: The following list shows the recognized application classes, along with a generic description of the valid metadata types available in each class:

X`00´ - X`04´
: Reserved.

X`05´
: Relational database data Allowed metadata types and their meaning are defined by the Distributed Relational Database Architecture (DRDA).

X`06´ - X`FF´
: Reserved.

REFTYP: This field specifies in what way the following parameter provides the described metadata value. Valid entries and their meanings are:

X`00´ No attribute parameter is provided. This is the default.

X`01´ An immediate constant is provided as a metadata value.

X`02´ - X`FF´
Reserved.

REFID: The content of this field is dependent on the REFTYP parameter.

REFTYP Field Content.

X`00´ Reserved.

X`01´ An unsigned one-byte binary number specifying an immediate constant.

X`02´ - X`FF´
Reserved.

The repeating group may occur zero, one, or more times. It is used to specify Subsetting Criteria for the tagged data if not all of the tagged data are to be annotated with the metadata, but only a particular subarray or an individual field.

Each group defines a criterion being associated with a specified dimension of the tagged data, by providing a low and a high index position limit. If no criterion is given for a dimension, then all positions in that dimension qualify. At most one criterion may be specified per dimension. If more than one specification occurs, an exception condition with exception-id 03 is given.

The repeating group parameters are:

CRITDIM: This parameter defines for which dimension the subsequent limit criteria are to be observed; 1 denotes the highest dimension, 2 the second highest, and so on. An exception condition with exception-id 07 arises if the specified number is greater than the number of dimensions present in the tagged data structure.

LOWLIM, HIGHLIM: If a field of the tagged data structure has, in the specified dimension, a position equal to or higher than the Low Index Limit and equal to or lower than the High Index Limit, then it is eligible for annotation.

A special rule applies if one or the other limit is specified as 0. If the Low Index Limit is 0, then it is treated as if it were equal to the High Index Limit. If the High Index Limit is 0, then it is treated as if it were equal to the highest existing position in this dimension (that is, the extent of the dimension).

If both are specified as 0, an exception condition with exception-id 03 is given.

Note: The above discussed dimensions are relative to the described data structure, which in turn may be part of a higher-level structure and thus may exist within higher-level dimensions. For example, if an RLO is tagged that describes a row, then it has only one dimension, and this is referred to by using 1 for denoting the highest and only dimension of this data structure. This does not preclude the RLO from being used and referenced by another, higher-level RLO that causes this row to become part of a two-dimensional structure.

Mapping of Metadata to the Data Being Annotated:

The metadata and the data being annotated with metadata may each be single items, or more or less regular arrays made from several items, perhaps with several dimensions.

If both are single items, then obviously the single metadata item is tagging the single data item.

If the metadata, or the tagged data, or both, are arrays, regular or not, then the mapping is done according to the following rule:

- The metadata is viewed, along the direction of its highest dimension, as a vector of *n* slices, *n* being 1 or more.

- The tagged data is viewed as a vector or sequence of *m* slices, along the direction of its highest dimension or, depending on the metadata type, along the direction of its second highest dimension; *m* may be 1 or more.
- Assignment of metadata structures to the tagged data structures is now done component-wise: the first slice of metadata to the first slice of the tagged data, the second slice of the metadata to the second slice of the tagged data, and so on. If *n* exceeds *m*, then the excessive metadata items are ignored; if *n* is less than *m*, then the trailing tagged data slices do not get any metadata slices assigned to them.

4.3.2 Supportive General-Purpose Triplets

This section describes supportive general-purpose triplets.

4.3.2.1 Continue Preceding Triplet (CPT)

This triplet logically continues the contents of a preceding triplet.

Syntax

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
0	UBIN	LENGTH	4 - 255	Length of triplet	M	N	X'02'
1	CODE	TYPEID	X'7F'	Triplet type ID: Continue Preceding Triplet	M	N	X'64'
2	UNDF	RES	X'00'	Reserved	M	N	X'02'
3 - 254	UNDF	CONTENT		Continued contents	M	N	X'26'

Semantics

The primary purpose of this triplet is to allow for a sequence of specifications or immediate data longer than coverable by a one-byte length field. It logically continues the specification of a physically preceding triplet. It must immediately follow the continued triplet.

All triplets which contain a repeating group at the end, if not otherwise indicated, can be continued with the CPT triplet.

A sequence of two or more Continue Preceding Triplet constructs may be used if necessary.

An exception condition with exception-id 13 exists if this triplet follows a triplet of a type other than those listed above.

RES is a reserved parameter and must contain a value of all-zero bits.

CONTENT contains an integral number of occurrences of the repeating group of the continued triplet. This parameter is interpreted as a logical continuation of a preceding triplet.

4.3.2.2 Implementation Support Data (ISD)

The Implementation Support Data triplet carries data which determines the required FD:OCA support for the Formatted Data Object.

Syntax

Offset	Type	Name	Range	Meaning	M/O	DEF	EXC
0	UBIN	LENGTH	5 - 6	Length of triplet	M	N	X'02'
1	CODE	TYPE	X'7E'	Triplet type ID: Implementation Support Data	M	N	X'44'
2	CODE	ID	X'00'	Construct identity	O	N	X'00'
3 - 4	CODE	SUBSET		Subset	M	N	X'06'
5	CODE	VERSION	X'01'	Architecture version	O	Y	X'02'

Semantics

ID is an unused parameter; it may contain a value of all-zero bits. The field exists only for the sake of syntactical uniformness among related constructs.

SUBSET identifies the highest architecture subset support required for this object. Valid entries and their meanings are:

X`0000´ FD:OCA Subset 0000 Base must be supported.

X`0100´ FD:OCA Subset 0100 Tower for DRDA support must be supported.

See Section 5.2 on page 73 for a definition of these subsets. An exception condition with exception-id 12 exists if an undefined subset is indicated, and the subset is assumed to be not supported.

VERSION identifies the version of the architecture used for the object. The value must be one. The default value is architecture version one. An exception condition with exception-id 12 exists if a version other than one is indicated. In such a case, version one is assumed.

The ISD triplet is required for all subsets of the architecture except the DRDA function set, Subset X'0100', and the ISD triplet must occur as the first triplet in the Descriptor. If an ISD occurs anywhere else in the Descriptor, it is ignored and an exception condition with exception-id 13 is raised.

If the ISD triplet is not specified, the DRDA support subset X'0100' is the default.

4.3.3 Registry of Data Types

This section formally describes the syntax and semantics of all FD:OCA registered data types and can be used by other architectures for reference purposes. The terms used below are defined as:

Field Type

Contains the hexadecimal code point to be used in the one-byte Field Type entry. In general there are two code points: one for a non-nullable type and one for a nullable type. A nullable field is prefixed with a null-indicator, which expresses whether or not the subsequent field has a value or actually is undefined. The null indicator is one byte long, precedes the field value immediately, and indicates in its high-order bit whether a value follows or not. If the high-order bit is B'1', then no value exists; if it is B'0', then a well-defined value is follows.

Parameters

As and when applicable, provide further qualification of the data type to be specified in the eight-byte Type Parameters field. Byte 0 refers to the leftmost byte of that field, byte 1 to the second-leftmost, and so on. All bytes not explicitly described are reserved and must have a value of all-zero bits.

Depending on the type, particular portions of the Type Parameter field are used to control the syntax and the semantics of the type. The type parameter Mode located in Byte 5, for example, is used to specify syntactic variants of some types. Thus for variable-length data types a mode bit controls the interpretation of the length field that precedes the data value. A B'0' defines that a non-zero field length value indicates the space reserved for data and that all space is transmitted whether it contains valid data or not. A B'1' shows that a non-zero field length value indicates the maximum value for the length field. Only enough space to contain each data value is transmitted.

Default Specifies a default value for the Type Parameters field, including any reserved parts.

Syntax Describes the inner structure of the value. For complex structures the value is represented by a sequence of characters, each character representing one byte of the value with the following meaning:

L Length field.

b Value byte.

N Null-byte (all-zero bits).

LLbb...bb would therefore represent a two-byte length field followed by several value bytes.

Semantics

Contains the interpretation rules for the value components defined in Syntax.

4.3.3.1 *String Data Types*

Byte String, Fixed Length, Field Type X'01'

Byte String, Fixed Length, Nullable, Field Type X'81'

Parameters	**Bytes 0-5**	Reserved.
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 0 - 32767

Default	X'0000000000000001'
Syntax	byte string
Semantics	None.

Byte String, Variable Length, Field Type X'02'

Byte String, Variable Length, Nullable, Field Type X'82'

Parameters	**Bytes 0-4**	Reserved.
	Byte 5	Mode format: bit string value: X'00' - X'01'
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 0 - 32767

Default	X'0000000000000000'
Syntax	*LLbbb...bb* is a two-byte signed binary integer followed by zero or more value bytes as defined below. If Field Length = 0: Length of *bbb...bb* is *LL*. If Field Length > 0: *LL* must be ≤ Field Length. If mode-bit 7 is B`1´: Length of *bbb...bb* is *LL*, and Field Length expresses the maximum value allowed for *LL*. If mode-bit 7 is B`0´: Length of *bbb...bb* is Field Length.
Semantics	The leftmost *LL* bytes of *bbb...bb* are the value.

Null-Terminated Byte String, Field Type X'03'

Null-Terminated Byte String, Nullable, Field Type X'83'

Parameters	**Bytes 0-4**	Reserved
	Byte 5	Mode format: bit string value: X'00' - X'01'
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 0 - 32767

Default	X'0000000000000000'
Syntax	*bbb...bbN* is zero or more non-zero bytes, followed by a byte of all-zero bits. If Field Length $k = 0$: The first occurrence, from left to right, of a byte with all-zero bits defines the actual length of the field. If Field Length has a value $k > 0$ then If mode-bit 7 is B`1´: $k$ is an upper limit for the number of value bytes; the leftmost all-zero byte defines the end of the field. If mode-bit 7 is B`0´: The field is k+1 bytes long, but an all-zero byte may occur earlier.
Semantics	Value is the leftmost string of non-zero bytes; it may be empty.

Short Byte string, Field Type X‘07’

Short Byte string, Nullable, Field Type X‘87’

Parameters	**Bytes 0-4**	Reserved.
	Byte 5	Mode format: bit string value: X‘00’ - X‘01’
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 0 - 255

Default	X‘0000000000000000’
Syntax	*Lbbb...bb* is a one-byte unsigned binary integer followed by zero or more value bytes as defined below. If Field Length = 0: Length of *bbb...bb* is *L*. If Field Length > 0: *L* must be ≤ Field Length. If Mode-Bit 7 is B`1´: Length of *bbb...bb* is *L*, and Field Length expresses the maximum value allowed for *L*. If Mode-Bit 7 is B`0´: Length of *bbb...bb* is Field Length.
Semantics	The leftmost *L* bytes of *bbb...bb* are the value.

4.3.3.2 *Character Data Types*

Character data implies that bit patterns are interpreted as strings of symbols, such as numeric digits, punctuation symbols, ideagrams, and so on. These symbols could, for example, come from a Cyrillic alphabet, or could be mathematical symbols, or perhaps be a mixture of Japanese Kanji and Latin alphabet characters. Thus, for a correct interpretation the reader needs to know which set of graphic characters is employed. Secondly, given the set of graphic characters, the reader needs to know what code page maps the characters to electronic bit patterns. Finally, if more than one code page and set of graphic characters occurs, the method must be known by which a switch from one code page and set of graphic characters to the next is indicated.

The first two of these essential parameters are sometimes represented through identifiers called:

1. Graphic Character Set Global Identifier (GCSGID)
2. Code Page Global Identifier (CPGID)

Products typically encode and combine them into a pair of two-byte binary numbers, called Coded Graphic Character Set Global Identifier (CGCSGID) or GCID for short.

The Character Data Representation Architecture (CDRA) defines the third essential parameter, the Encoding Scheme Identifier, abbreviated ESID. CDRA also defines how the ESID, together with one or more pairs of GCSGID and CPGID and other coding-related information, allows an unambiguous interpretation of the many character-string encoding methods that are in common use around the world.

The combination of ESID and the other associated identifiers can be referred to by a short name, a 16-bit identifier called Coded Character Set Identifier (CCSID). CDRA defines how a CCSID points to a detailed and long-form description of the above discussed parameters. CDRA also has registered a great number of very common combinations of ESID and GCSGID/CPGID pairs, and then has assigned a CCSID to each of them.

FD:OCA uses the 16-bit CCSID in its Character Data Type parameters. Normally, this will be a registered CCSID, and therefore the implied ESID and GCSGID/CPGID pairs are known. In the less frequent cases of non-registered CCSIDs, the CCSID-Resource construct carries the actual information, and is referenced through the CCSID value in the type parameters.

For migration purposes, FD:OCA in its current version also supports the traditional CGCSGID concept, but only in the simple form, where a single CGCSGID is needed to describe a character string. Strings requiring more than one character set and code page must be described through the CCSID method.

Coexistence of CCSIDs and CGCSGIDs is possible through the following convention. A four-byte long area is reserved in all the type parameters, for the character-string encoding information. This area either carries a CGCSGID, which is a pair of two-byte binary numbers, or 16 bits of zeros followed by a CCSID, which is a 16-bit identifier. Thus, a CCSID in FD:OCA type parameters is always preceded by 16 bits of zeros. The subsequent character data type descriptions will not specifically mention CGCSGIDs, but wherever a CCSID is placed, a CGCSGID would also fit and is allowed.

Common Default Rules for Character Data

The subsequent diagrams define several variants of character data. If some or all of the relevant type parameters are left unspecified, then the following default rules apply:

Table 4-3 Default Rules for Character Data

Byte 0-3	If byte 4 is X'01' or unspecified, the default is X'000001F4'; that is, CCSID X'01F4'. If byte 4 is X'02', the default is X'0000112C'; that is, CCSID X'112C'.
Byte 4	The default will be derived from byte 0-3. It is the number of bytes per character code point expressed as an eight-bit unsigned binary number, if all characters are encoded with the same number of bytes, else it is X'01'.
Byte 5	The default is X'00'.
Byte 6-7 (normal form)	For the fixed-length types the default is X'0001'. For the variable-length types and null-terminated types the default is X'0000'.
Byte 6-9 (long form)	For the fixed-length types the default is X'00000001'. For the variable-length types and null-terminated types the default is X'00000000'.
Byte 6-13 (very long form)	For the fixed-length types the default is X'0000000000000001'. For the variable-length types the default is X'0000000000000000'.

If the value in bytes 0-3 consists of all one-bits, then the environment carrying the FD:OCA object may determine the actual CCSID. Any value existing already in byte 4 is ignored in this case. If the environment cannot determine a valid CCSID, then the FD:OCA-defined defaults are used.

Character String, Fixed Length, Field Type X'10'

Character String, Fixed Length, Nullable, Field Type X'90'

Parameters	**Bytes 0-3**	CCSID
	Byte 4	Character-Length Identifier format: unsigned binary integer units: bytes value: 1 for SBCS and mixed SBCS/DBCS 2 for DBCS
	Byte 5	Reserved.
	Bytes 6-7	Field Length format: signed binary integer units: characters for DBCS; else bytes value: 0 - 32767 for SBCS and mixed SBCS/DBCS 0 - 16383 for DBCS

Default	See Table 4-3 on page 43.
Syntax	Byte string
Semantics	CCSID defines semantics.

Character String, Variable Length, Field Type X'11'

Character String, Variable Length, Nullable, Field Type X'91'

Parameters	**Bytes 0-3**	CCSID
	Byte 4	Character-Length Identifier format: unsigned binary integer units: bytes value: 1 for SBCS and mixed SBCS/DBCS 2 for DBCS
	Byte 5	Mode format: bit string value: X'00' - X'01'
	Bytes 6-7	Field Length format: signed binary integer units: characters for DBCS; else bytes value: 0 - 32767 for SBCS and mixed SBCS/DBCS 0 - 16383 for DBCS

Default	See Table 4-3 on page 43.
Syntax	*LLbbb...bb* is a two-byte signed binary integer followed by zero or more value bytes as defined below. If Field Length = 0: Length of *bbb...bb* is *LL* times Character-Length Identifier. If Field Length > 0: *LL* must be ≤ Field Length. If mode-bit 7 is B`1´: Length of *bbb...bb* is *LL* times Character-Length Identifier; Field Length expresses the maximum value allowed for *LL*. If mode-bit 7 is B`0´: Length of *bbb...bb* is Field Length times Character-Length Identifier.
Semantics	Value follows the *LL* bytes, unless *LL* is zero. The length of the value in bytes is *LL* times Character-Length Identifier. The CCSID defines the semantics of the *LL* characters.

Null-Terminated Character String, Field Type X'14'

Null-Terminated Character String, Nullable, Field Type X'94'

Parameters	**Bytes 0-3**	CCSID
	Byte 4	Character-Length Identifier format: unsigned binary integer units: bytes value: 1 for SBCS and mixed SBCS/DBCS 2 for DBCS
	Byte 5	Mode format: bit string value: X'00' - X'01'
	Bytes 6-7	Field Length format: signed binary integer units: characters for DBCS; else bytes value: 0 - 32767 for SBCS and mixed SBCS/DBCS 0 - 16383 for DBCS

Default	See Table 4-3 on page 43.
Syntax	*bbb...bbN* is zero or more value bytes, as defined below, followed by a byte of all-zero bits. If Field Length k= 0: The first occurrence, from left to right, of a byte with all-zero bits defines the length of the field. If Field Length has a value $k > 0$ and m is k times Character Length Identifier, then: If mode-bit 7 is B`1´: $m$ is an upper limit for the number of value bytes; the leftmost all-zero byte defines the actual end of the field; If mode-bit 7 is B`0´: The field is m+1 bytes long, but an all-zero byte may occur earlier.
Semantics	Value is the leftmost string of non-zero bytes; it may be empty. The CCSID defines the semantics of the value bytes.

Short Character String, Variable Length, Field Type X'19'

Short Character String, Variable Length, Nullable, Field Type X'99'

Parameters	Bytes 0-3	CCSID
	Byte 4	Character-Length Identifier format: unsigned binary integer units: bytes value: 1 for SBCS and mixed SBCS/DBCS 2 for DBCS
	Byte 5	Mode format: bit string value: X'00' - X'01'
	Bytes 6-7	Field Length format: signed binary integer units: characters for DBCS; else bytes value: 0 - 255 for SBCS and mixed SBCS/DBCS 0 - 127 for DBCS

Default	See Table 4-3 on page 43.
Syntax	*Lbbb...bb* is a one-byte unsigned binary integer followed by zero or more value bytes as defined below. If Field Length = 0: Length of *bbb...bb* is *L* times Character Length Identifier. If Field Length > 0: *L* must be ≤ Field Length. If mode-bit 7 is B`1´: Length of *bbb...bb* is *L* times Character-Length Identifier; Field Length expresses the maximum value allowed for *L*. If mode-bit 7 is B`0´: Length of *bbb...bb* is Field Length times Character-Length Identifier.
Semantics	Value follows the *L* byte, unless *L* is zero. The length of the value in bytes is *L* times Character-Length Identifier. The CCSID defines the semantics of the *L* characters.

Numeric Character Strings

A certain subset of the data type Character Data, called Numeric Character Strings, is defined below. Values of this type may be useful to bridge the gap between text and numeric data. While they are character strings, they may be converted into numbers.

Numeric character strings are values of the type Character Data, with the following syntax restrictions and semantics.

The allowed sequence of characters is:

```
SI ...  IDF ...  FX
```

where:

S	is an optional arithmetic character Plus Sign or Typographic Minus Sign. If it occurs, it is the first character.
I ... I	is a sequence of the numeric decimal characters One, Two, Three, Four, Five, Six, Seven, Eight, Nine, Zero.
D	is a single optional punctuation character Comma, Colon or Period. This character is restricted to 0 or 1 occurrences.
F ... F	is a sequence of the numeric decimal characters One, Two, Three, Four, Five, Six, Seven, Eight, Nine, Zero.
X	is an optional component appropriate for representation of floating point numbers. It consists of the following components:

```
*BB**ZE ... E
```

where:

*	must occur and must be the special character Asterisk.
BB	must occur and is one of the following sequences of decimal characters: Two, One Zero, One Six.
**	must occur and must be a sequence of two Asterisk characters.
Z	is an optional arithmetic character Plus Sign or Typographic Minus Sign.
E ... E	must occur and is a sequence of the numeric decimal characters One, Two, Three, Four, Five, Six, Seven, Eight, Nine, Zero.

The occurrence of *I ... I* and *F ... F* is optional, but at least one character must occur. If *F ... F* occurs, *D* must also occur.

Semantics

For a string *SIIIIDFFFF*BB**ZEE* the value is the following decimal number: *S(IIII plus 0.FFFF)* multiplied by *BB* to the power of *Z(EE)*. Missing IIII or FFFF is equivalent to a value of 0 for either of these components.

In other words, simple decimal numbers in the form of character strings are recognized, as well as character strings representing decimal numbers in a so-called scientific notation, with a base and exponent value; three different bases, namely 2, 10, and 16, may be used.

4.3.3.3 *Numeric Data Types*

This section describes integer data types, fixed point data types, and floating point data types.

Integer Data Types

This section describes integer data types.

Unsigned Binary Integer, Field Type X'22'

Unsigned Binary Integer, Nullable, Field Type X'A2'

Parameters	**Bytes 0-5**	Reserved.
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 1, 2, 4, 8

Default	X'0000000000000004'
Semantics	If the field consists of n bits, $bn \mid ... \mid b2 \mid b1$, then its value is sum ($bi \times 2^{i-1}$).

Signed Binary Integer, Field Type X'23'

Signed Binary Integer, Nullable, Field Type X'A3'

Parameters	**Bytes 0-5**	Reserved.
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 1, 2, 4, 8

Default	X'0000000000000004'
Syntax	The first bit is the sign bit (B'0' = positive; B'1' = negative).
Semantics	Positive numbers are in true binary notation; negative numbers are in two's complement notation.

PC(8087) Signed Binary Integer, Field Type X'24'

PC(8087) Signed Binary Integer, Nullable, Field Type X'A4'

Parameters	**Bytes 0-5**	Reserved.
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 1, 2, 4, 8

Default	X'0000000000000004'
Syntax	*SBn* | ... | *B*0 The first bit (*S*) is the sign bit (B'0' = positive; B'1' = negative). The bits following the sign bit (*Bn*...*B0*) represent a binary number. The data occurs in the data stream in byte reversed order; that is, the sign is located in the highest addressed, or the rightmost byte, preceded by the byte containing bits *Bn*-1. The leftmost byte is the byte containing bit *B*0.
Semantics	Positive numbers are in true binary notation; negative numbers are in two's complement notation. Zero is represented with all bits zero (*S* = positive).

Fixed Point Data Types

This section describes fixed point data types.

Decimal Fixed Point, Field Type X'30'

Decimal Fixed Point, Nullable, Field Type X'B0'

Parameters	**Bytes 0-4**	Reserved.
	Byte 5	Mode format: bit string value: X'00' - X'01'
	Byte 6	Field Length format: signed binary integer units: digits value: 1 - 31
	Byte 7	Number of Fractional Digits format: signed binary integer units: digits value: –128 to +127

<table>
<tr><td>Default</td><td>X‘0000000000000802’</td></tr>
<tr><td>Syntax</td><td>There are two variants, a signed and an unsigned version, depending on the Mode flag:

Mode X`00´: Every half-byte contains a binary encoded decimal digit; the last half-byte contains the sign <i>DD</i>|<i>DD</i>|...|<i>DS</i>, where <i>D</i> represents a digit, and <i>S</i> the sign.<br>Mode X`01´: Every half-byte contains a binary encoded decimal digit, including the last one. A positive sign is implied. DD|DD|...|DD

A leading unused half-byte must contain zero.

The encoding of digits and signs is as follows:
<pre>
Encoding   Digits   Sign

B‘0000’   ‘zero’   Invalid
B‘0001’   ‘one’   Invalid
B‘0010’   ‘two’   Invalid
B‘0011’   ‘three’ Invalid
B‘0100’   ‘four’   Invalid
B‘0101’   ‘five’   Invalid
B‘0110’   ‘six’   Invalid
B‘0111’   ‘seven’ Invalid
B‘1000’   ‘eight’ Invalid
B‘1001’   ‘nine’   Invalid
B‘1010’   Invalid ‘plus sign’
B‘1011’   Invalid ‘minus sign’
B‘1100’   Invalid ‘plus sign’
B‘1101’   Invalid ‘minus sign’
B‘1110’   Invalid ‘plus sign’
B‘1111’   Invalid ‘plus sign’
</pre></td></tr>
<tr><td>Semantics</td><td>Let the stored value be val, then the semantics are $val \times 10^{-\text{(number of fractional digits)}}$.</td></tr>
</table>

Unsigned Binary Fixed Point, Field Type X'34'

Unsigned Binary Fixed Point, Nullable, Field Type X'B4'

Parameters	**Bytes 0-4**	Reserved.
	Byte 5	Mode format: bit string value: X'00' - X'02'
	Byte 6	Field Length format: signed binary integer units: depending on the Mode flag: Mode X`00´: bytes Mode X`01´: bytes Mode X`02´: decimal digits value: depending on the Mode flag:  Mode X`00´: 2, 4, 8 Mode X`01´: 2, 4, 8 Mode X`02´: 1 to 18
	Byte 7	Number of Fractional Digits format: signed binary integer units: digits value: –128 to +127

Default	X'0000000000000400'
Syntax	unsigned binary integer **Note:** If the Field Length is specified in decimal digits (Mode X'02'), the actual size of the binary integer is for: 1 - 4 digits 2 bytes 5 - 9 digits 4 bytes 10 - 18 digits 8 bytes
Semantics	Let the stored value be val, then the semantics are, depending on the Mode flag: Mode X`00´: $val \times 2^{-\text{(number of fractional digits)}}$ Mode X`01´: $val \times 10^{-\text{(number of fractional digits)}}$ Mode X`02´: $val \times 10^{-\text{(number of fractional digits)}}$

Signed Binary Fixed Point, Field Type X'31'

Signed Binary Fixed Point, Nullable, Field Type X'B1'

Parameters	Bytes 0-4	Reserved.
	Byte 5	Mode format: bit string value: X'00' - X'02'
	Byte 6	Field Length format: signed binary integer units: depending on the Mode flag: Mode X`00´: bytes Mode X`01´: bytes Mode X`02´: decimal digits value: depending on the Mode flag: Mode X`00´: 2, 4, 8 Mode X`01´: 2, 4, 8 Mode X`02´: 1 to 18
	Byte 7	Number of Fractional Digits format: signed binary integer units: digits value: –128 to +127

Default	X'0000000000000400'
Syntax	signed binary integer **Note:** If the Field Length is specified in decimal digits (Mode X'02'), the actual size of the binary integer is for: 1 - 4 digits 2 bytes 5 - 9 digits 4 bytes 10 - 18 digits 8 bytes
Semantics	Let the stored value be val, then the semantics are, depending on the Mode flag: Mode X`00´: $val \times 2^{-\text{(number of fractional digits)}}$ Mode X`01´: $val \times 10^{-\text{(number of fractional digits)}}$ Mode X`02´: $val \times 10^{-\text{(number of fractional digits)}}$

Fixed Point Numeric Character String, Field Type X'32'

Fixed Point Numeric Character String, Nullable, Field Type X'B2'

Parameters	**Bytes 0-3**	CCSID
	Byte 4	Digit Length Indicator format: unsigned binary integer units: bytes value: 1 - 2
	Byte 5	Mode format: bit string value: X'00' - X'02'
	Byte 6	Field Length format: signed binary integer units: digits value: 1 - 31
	Byte 7	Number of Fractional Digits format: unsigned binary integer units: digits value: 1 - 31

Default	X'02B901F401000800'
Syntax	A string of numeric characters, 0 through 9, along with an optional sign byte, not counted in the field length, containing a +, –, or space character. The presence and position of the sign byte is indicated through the Mode flag, as follows: Mode X`00´: The sign byte precedes the string. Mode X`01´: The sign byte follows the string. Mode X`02´: No sign byte is present. The number of fractional digits must be ≤ Field Length.
Semantics	The sequence of digits represents a value *val*. *val* is a positive decimal integer if the sign byte is + or space character or missing. *val* is a negative decimal integer if the sign byte is –. The semantics of value *val* are $val \times 10^{-(\text{number of fractional digits})}$.

Zoned Decimal Fixed Point, Field Type X'33'

Zoned Decimal Fixed Point, Nullable, Field Type X'B3'

Parameters	**Bytes 0-4**	Reserved.
	Byte 5	Mode format: bit string value: X'00' - X'01'
	Byte 6	Field Length format: signed binary integer units: digits value: 1 - 31
	Byte 7	Number of Fractional Digits format: unsigned binary integer units: digits value: 1 - 31

Default	X‘0000000000000800’
Syntax	Every byte contains a left half-byte, called zone (Z), and a right half-byte which is a binary encoded decimal digit (D). The zone of the last or the first byte, depending on the Mode, is an arithmetic sign: Mode X`00´: Left nibble of last byte is the sign. Mode X`01´: Left nibble of first byte is the sign. The encoding of digits and signs is as follows: `Encoding  Digits   Sign` `B‘0000’   ‘zero’   Invalid` `B‘0001’   ‘one’    Invalid` `B‘0010’   ‘two’    Invalid` `B‘0011’   ‘three’  Invalid` `B‘0100’   ‘four’   Invalid` `B‘0101’   ‘five’   Invalid` `B‘0110’   ‘six’    Invalid` `B‘0111’   ‘seven’  Invalid` `B‘1000’   ‘eight’  Invalid` `B‘1001’   ‘nine’   Invalid` `B‘1010’   Invalid  ‘plus sign’` `B‘1011’   Invalid  ‘minus sign’` `B‘1100’   Invalid  ‘plus sign’` `B‘1101’   Invalid  ‘minus sign’` `B‘1110’   Invalid  ‘plus sign’` `B‘1111’   Invalid  ‘plus sign’` The encoding of the zone is B‘1111’ (X‘F’). Number of fractional digits must be ≤ Field Length.
Semantics	If *val* is a positive or negative integer as represented by the decimal digits *DD...D* and the sign *S*, then the value is $val \times 10^{-(\text{number of fractional digits})}$.

COBOL/2 Zoned Decimal Fixed Point, Nullable, Field Type X'35'

COBOL/2 Zoned Decimal Fixed Point, Nullable, Field Type X'B5'

Parameters	**Bytes 0-4**	Reserved.
	Byte 5	Mode format: bit string value: X'00' - X'01'
	Byte 6	Field Length format: signed binary integer units: digits value: 1 - 31
	Byte 7	Number of Fractional Digits format: unsigned binary integer units: digits value: 1 - 31

Default	X‘0000000000000800’
Syntax	Every byte contains a left half-byte, called zone (*Z*), and a right half-byte which is a binary encoded decimal digit (*D*). The zone of the last or the first byte, depending on the Mode, is an arithmetic sign: Mode X`00´: Left nibble of last byte is the sign. Mode X`01´: Left nibble of first byte is the sign. The encoding of digits and signs is as follows: Encoding Digits Sign B‘0000’ ‘zero’ ‘plus sign’ B‘0001’ ‘one’ ‘plus sign’ B‘0010’ ‘two’ ‘plus sign’ B‘0011 ‘three’ ‘plus sign’ B‘0100 ‘four’ ‘minus sign’ B‘0101 ‘five’ ‘minus sign’ B‘0110 ‘six’ ‘minus sign’ B‘0111 ‘seven’ ‘minus sign’ B‘1000 ‘eight’ ‘plus sign’ B‘1001 ‘nine’ ‘plus sign’ B‘1010 Invalid ‘plus sign’ B‘1011 Invalid ‘plus sign’ B‘1100 Invalid ‘minus sign’ B‘1101 Invalid ‘minus sign’ B‘1110 Invalid ‘minus sign’ B‘1111 Invalid ‘minus sign’ The encoding of the zone is B‘0011’ (X‘3’). Number of fractional digits must be ≤ Field Length.
Semantics	If *val* is a positive or negative integer as represented by the decimal digits *DD...D* and the sign *S*, then the value is $val \times 10^{-(\text{number of fractional digits})}$.

Floating Point Data Types

Floating point numbers are a subset of the rational numbers; common terms to describe their syntax and semantics are:

- **Syntactical Definitions**

  Figure 4-1 shows the value structure of the floating point data types discussed here:

sign	characteristic	fraction

Figure 4-1 Structure of a Floating Point Number

sign (*S*) Usually the leftmost value bit.

B'0' = positive value

B'1' = negative value

characteristic (*e*)
Unsigned binary or decimal integer, used to determine the exponent.

fraction Binary, decimal, or hexadecimal number that determines the significand.

- **Semantic Definitions**

base (*B*) Specifies number system and base for the exponentiation.

bias The number which adjusts the characteristic to get the exponent.

exponent (*E*)
Characteristic minus bias, E = e – bias.

significand (*M*)
Derived from the fraction by interpreting the fraction in the number system given by the base. The exact rules are type-specific.

The above components constitute the semantic value of a floating point number as $(-1)^S \times M \times B^E$.

Hexadecimal Floating Point, Field Type X'40'

Hexadecimal Floating Point, Nullable, Field Type X'C0'

Parameters	**Bytes 0-5**	Reserved.
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 4, 8, 16

Default	X'0000000000000008'

Syntax	Structure: Depending on the Field Length, several formats are possible:
	
	`format      length  charact. fraction`
	`-------------------------------------`
	`short        4 bytes  7 bits    6 hex`
	`long         8 bytes  7 bits   14 hex`
	`extended    16 bytes  7 bits   28 hex`
	
	For extended format see Note below.

Semantics	For fraction $h1 \mid ... \mid hn$, the significand is $0.h1 \mid ... \mid hn$.
	
	Base: 16
	bias : Depending on the format, this is:
	
	`format      bias`
	`----------------`
	`short         64`
	`long          64`
	`extended      64`
	Normalized numbers:
	$0 <$ characteristic $\leq$ maximum and $h1 \neq 0$
	Denormalized numbers:
	characteristic = 0 , and fraction $\neq 0$
	Maximum characteristic:
	127
	Sign: Bit is B'0' for positive values, and B'1' for negative values.
	Value of zero: characteristic = 0 and fraction = 0, sign=B'0' or B'1'
	Note: For the extended form, the content of the ninth byte is ignored and the remaining seven bytes are thought of as following the eighth byte.

Binary Floating Point (IEEE et al.[1]), Field Type X'47' and X'48'

Binary Floating Point (IEEE et al.[2]), Nullable, Field Type X'C7' and X'C8'

Parameters	**Bytes 0-1**	Reserved.
	Bytes 2-3	Bias indicator format: unsigned binary integer value: 0 or 1
	Bytes 4-5	Reserved.
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 4, 8, 16

IEEE ANSI/IEEE Std. 745-1985, Binary Floating Point Arithmetic. See also Note.

<table>
<tr><td>Default</td><td>X‘0000000000000004’</td></tr>
<tr><td>Syntax</td><td>Structure: Depending on the Field Length, several formats are possible:
<pre>
format   length   charact.   fraction
----
short   4 bytes   8 bits   23 bits
long   8 bytes   11 bits   52 bits
extended 16 bytes   15 bits   112 bits
</pre>
Types X`47´ and X`C7´:
The data occur in the data stream in byte-reversed order; that is, the rightmost byte has the lowest address, and the leftmost has the highest address.
Types X`48´ and X`C8´:
Byte addresses increase from left to right.</td></tr>
<tr><td>Semantics</td><td>Base: 2
Significand: For the fraction $b1|...|bn$, the significand depends on the characteristic:
<pre>
IF the characteristic is not zero
THEN the significand is   1.b1|...|bn;
ELSE it is   0.b1|...|bn.
</pre>
Maximum characteristic and bias:
Depending on format and bias indicator, these are:
<pre>
        bias indicator 0 bias indicator 1
format max.char.   bias max.char.   bias
----
short   254   127   255   128
long   2046   1023   2047   1024
extended   32766 16383   32767 16384
</pre>
Sign: Bit is B‘0’ for positive values, and B‘1’ for negative values.
Value of zero: characteristic = 0 and fraction = 0, sign=B‘0’ or B‘1’
Note: These types cover a class of several representations of binary floating point numbers. The IEEE Binary Floating Point Numbers are the subset with bias indicator 0 and short or long format. Types X‘47’ and X‘C7’ indicate byte-reversed storage; hence they cover the PC variants of these numbers, while types X‘48’ and X‘C8’ are for System/390-style data streams.</td></tr>
</table>

VAX Binary Floating Point, Field Type X'49'

VAX Binary Floating Point, Nullable, Field Type X'C9'

Parameters	**Bytes 0-4**	Reserved.
	Byte 5	Mode format: bit string value: X'00' - X'01'
	Bytes 6-7	Field Length format: signed binary integer units: bytes value: 4, 8, 16

Default	X'0000000000000004'

Syntax	Structure: Depending on the Field Length and the Mode, several formats are possible:
	`format    length   charact. fraction mode`
	`----------------------------------------`
	`short      4 bytes  8 bits   23 bits X'00'(F-float)`
	`long wide  8 bytes  8 bits   55 bits X'01'(D-float)`
	`long       8 bytes 11 bits   52 bits X'00'(G-float)`
	`extended  16 bytes 15 bits  112 bits X'00'(H-float)`
	The data occurs in the data stream in byte-pairs with the leftmost byte-pair at the lowest address, and the rightmost at the highest address. The order of bytes within each byte-pair is reversed.

Semantics	Base: 2
	Significand: For the fraction $b1\|...\|bn$, the significand depends on the characteristic:
	`IF the characteristic is not zero`
	`THEN the significand is     0.1|b1|...|bn;`
	`ELSE it is                  0.0|b1|...|bn.`
	Maximum characteristic and bias:
	Depending on the format, these are:
	`format     max.char.   bias`
	`---------------------------`
	`short           255     128`
	`long wide       255     128`
	`long           2047    1024`
	`extended      32767   16384`
	Sign: Bit is B'0' for positive values, and B'1' for negative values.
	Value of zero: characteristic = 0 and sign = B'0'
	Reserved: characteristic = 0 and sign = B'1'

4.4 FD:OCA Data Component Content

The FD:OCA data component contains the value or sequence of values of a Formatted Data Object. The surrounding architecture may partition the data in one or more Data Structured Fields on a byte boundary.

There is no architecture regulated meaning connected with this data other than that expressed in the Descriptor.

4.5 Error Handling

This section outlines the error situations that may occur when parsing an FD:OCA Descriptor and how they are reported.

4.5.1 Exception Conditions

The information found in an FD:OCA descriptor or data part may be erroneous or invalid. Such a situation is called an exception condition. FD:OCA defines distinct identifiers, called exception-ids, for various exception conditions.

Exception conditions may be of syntactical nature; for instance, when a triplet appears in a form different from what is prescribed in this document. Or they may be of semantic nature, such as when a Decimal Fixed Point number is specified, but the field contains invalid digits. Some exception conditions may be associated with an individual triplet; others may pertain to a whole object, not any particular part of it.

Consequently, the description of exception conditions and their exception-ids is found in two different places in this document:

- The triplet-specific exception conditions and their exception-ids are defined in the text for each individual triplet specification.
- Certain common exception conditions (general syntax exceptions) can occur in any of the triplets. The possibilities for such errors are indicated in the syntax diagram of each triplet. Their definition and exception-ids are presented below, followed by the exception definitions and exception-ids for object-related errors.

In most cases, the definition spells out that the value of the affected object is undefined and thus can normally not be used; for some exception conditions, alternate or substitute values are prescribed. In any case, a product that accepts an FD:OCA object must detect any existing exception conditions in the object and handle them as specified here.

4.5.1.1 General Syntax Exceptions

The syntax description of each triplet indicates, in the column labeled EXC, what general syntax errors may be expected for any particular triplet parameter.

These general syntax exception categories are indicated in the following way:

		Code	
Position	**Exception Condition Category**	**Binary**	**Hexadecimal**
Bit 0	- - Reserved - - not used in FD:OCA	B′10000000′	X′80′
Bit 1	Construct type code ID not recognized	B′01000000′	X′40′
Bit 2	State or sequence violation	B′00100000′	X′20′
Bit 3	- - Reserved - - not used in FD:OCA	B′00010000′	X′10′
Bit 4	- - Reserved - - not used in FD:OCA	B′00001000′	X′08′
Bit 5	Missing mandatory parameter or parameter group	B′00000100′	X′04′
Bit 6	Parameter value not acceptable	B′00000010′	X′02′
Bit 7	- - Reserved - - not used in FD:OCA	B′00000001′	X′01′
None.	None.	B′00000000′	X′00′

The general syntax exception conditions are described by category as follows:

Exception-id 02
Construct type ID not recognized

An unknown construct type was detected; the construct cannot be identified. The value of the complete object is undefined.

Exception-id 03
State or sequence violation

A specification was found that is invalid under the particular circumstances or in the particular position, where it was found. A typical example is a reference to another construct that does not exist.

If this error exists in an attribute triplet, then the affected and any subsequent parts of the object are undefined.

Exception-id 06
Missing mandatory parameter or parameter group

A parameter or group of parameters is missing, although it is mandatory under the current circumstances.

If this error exists in an attribute triplet, then the affected and any subsequent parts of the object are undefined.

Exception-id 07
Parameter value not acceptable

A parameter value was found to be less than the prescribed minimum or more than the allowed maximum, or impossible in the current context.

If this error exists in an attribute triplet and if no default value for this parameter is defined, then the affected and any subsequent parts of the object are undefined.

If this error exists in an attribute triplet, but a default value for this parameter is defined, then this default value will be used.

The column labeled EXC of each triplet's syntax description has an eight-bit hexadecimal value, indicating what exception categories are to be expected for each operand of the triplet. If the first or leftmost bit of this value is a one, then exception condition with exception-id 1 is expected; the second leftmost bit corresponds to exception-id 2; the third to exception-id 3; and so on. Thus, a value of X'26' indicates that exception conditions with exception-ids 3, 6, and 7 are possible, and need to be checked.

4.5.1.2 *Object-Related Exceptions*

The following object-related exception conditions can be received:

Exception-id 80
Descriptor missing

If the FD:OCA object contains only values without attribute triplets, this exception condition exists. The value of the object is undefined.

Exception-id 84
Referenced object not found

If the object refers to another object that cannot be located, this exception condition exists. The value of the referenced object is undefined.

Exception-id 85
Attribute/Value mismatch

If the described value does not have the characteristics claimed by its attributes, this exception condition exists. In this case, the affected and any subsequent parts of the object are undefined.

Exception-id 86

More than one major attribute

If more than one major attribute exists within a Formatted Data Object, this exception condition exists. In such a case, the value of the Formatted Data Object is undefined. For a discussion on major attributes, see Section 4.3 on page 28.

4.5.1.3 Exception Reporting

If an exception condition has been detected, there may or may not be a need to communicate the error. If the exception condition needs to be communicated, FD:OCA defines in what form it is communicated.

In principle an exception condition will be described by its identifier and by pointers to the descriptor and data parts where the error has been detected. For this purpose, FD:OCA defines a block of 16 bytes called an exception reporting structure. Each exception reporting structure refers to a single triplet and the data described with the triplet. It can refer to a triplet where an exception condition has been detected, or a triplet which references, directly or indirectly, a triplet where an exception condition has been detected. If more than one exception reporting structure is used, the sequence of these structures should correspond to the relative sequence of the referenced triplets in the descriptor. FD:OCA considers the complete logical descriptor and the complete data as a physical string and ignores all bytes used for the Structured Field Introducers. FD:OCA assumes that the receiver of this exception description has the ability to select appropriate bytes from this object for an error analysis, if necessary. An exception reporting structure contains the following information:

Table 4-4 Exception Reporting Structure

Byte 0	If bytes 4-7 reference a triplet where an exception condition has been detected, this byte contains the FD:OCA-defined exception-id. If bytes 4-7 reference a triplet where a new exception condition has not been detected, but which references, directly or indirectly, a triplet where an exception condition has been detected, this byte contains all-zero bits.
Byte 1	Bit 7 of this byte tells whether a single exception reporting structure or a sequence of structures is used. Bit 7 of this byte is set to B'0' for the last or only exception reporting structure, and it is set to B'1' if additional structures follow. All other bits are not in use and must have a value of B'0'.
Bytes 2-3	Reserved, and must have all-zero bits.
Bytes 4-7	Specify the offset, with an unsigned binary integer, of the triplet where the error has been detected. For this purpose the complete logical descriptor is considered as a single physical string, independent of how the descriptor is represented by the implementing product. All Structured Field Introducer bytes are ignored. If no offset can be specified, the bytes must have all-one bits.
Bytes 8-9	Contain the offset, an unsigned binary integer, of the parameter in the logical triplet where the error has been detected. The offset is relative to the beginning of the triplet, except if the error has been detected in a CPT. In this case the offset is relative to the beginning of the continued triplet. If no offset can be specified, the bytes must have all-one bits.
Bytes 10-11	Reserved, and must have all-zero bits.
Bytes 12-15	The data portion described by the triplet referenced in bytes 4-7 is pointed to from this entry. Bytes 12-15 specify the offset, through an unsigned binary integer, of the affected data. For this purpose the complete data is considered as a single physical string, independently of how the data is represented by the implementing product. All Structured Field Introducer bytes are ignored. If no offset can be specified, the bytes must have all-one bits.

4.5.2 Exception IDs

The following is a list of all exception-ids that are used within the architecture:

- General Syntax Exceptions

  02 Construct type code not recognized

  03 State or sequence violation

  A specification is invalid in the particular context or position, such as a reference with a triplet identifier that does not exist.

  06 Missing mandatory parameter or parameter group

  07 Parameter value not acceptable

  A parameter value is outside the permitted value range or is impossible in the current context.

- Triplet-Specific Exceptions

  09 Reference to or from an invalid triplet type

  10 Dimension error

  More values than dimensions occur or the dimension specification is incorrect (for example, negative or zero).

  12 Source value outside target range

  It can therefore be predicted that the target value will be undefined.

  13 Triplet occurs too often or at the wrong position

- Object-Related Exceptions

  80 The FD:OCA object contains values only

  84 The referenced object cannot be located

  85 The described value does not have the characteristics described by its attributes

  86 More than one major attribute is specified in the FD:OCA object.

Chapter 5

Compliance

This chapter:

- Identifies the FD:OCA subsets
- Outlines the FD:OCA compliance rules
- Shows the code point assignments for the FD:OCA attribute triplets

5.1 FD:OCA Version

The architecture described in this document is FD:OCA Version 1.

5.2 FD:OCA Subsets

The descriptive facilities of FD:OCA are divided into a base set and towers. The towers in turn may be subdivided into nested subset levels, as illustrated in Figure 5-1. Products supporting FD:OCA must support all the functions of the base set, and may support some or all of the facilities of a tower. To support a tower with subsets, a product must select a tower and a subset level in that tower. The product must then support all the facilities of the selected tower subset and of any lower-level subsets, in addition to the facilities of the base set.

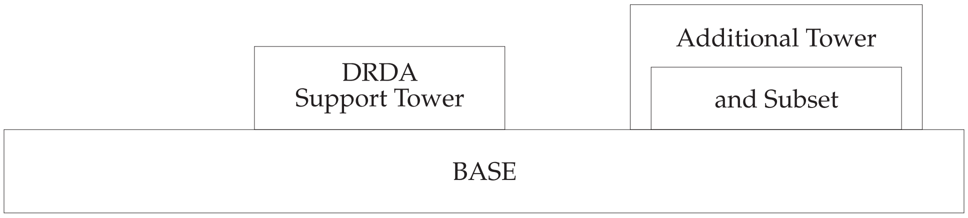

Figure 5-1 FD:OCA Base and Towers Concept

The following section defines the base set and any towers. For each functional subset, the valid constructs, parameter values, and combinations thereof are listed.

- FD:OCA Subset 0000, Base
  - — Attribute Triplets:
    - — Group Data Array (GDA) and Nullable Group Data Array
    - — Row Layout (RLO)
    - — Simple Data Array (SDA)
      - — Field Types:
        - — Binary Floating Point (IEEE et al.), Field Type X‘47’
          - — Bias Indicator 0

- Binary Floating Point (IEEE et al.), Nullable, Field Type X'C7'
  - Bias Indicator 0
- Binary Floating Point (IEEE et al.), Field Type X'48'
  - Bias Indicator 0
- Binary Floating Point (IEEE et al.), Nullable, Field Type X'C8'
  - Bias Indicator 0
- Byte String, Fixed Length, Field Type X'01'
- Byte String, Fixed Length, Nullable, Field Type X'81'
- Byte String, Variable Length, Field Type X'02'
  - Mode X'01'
- Byte String, Variable Length, Nullable, Field Type X'82'
  - Mode X'01'
- Character String, Fixed Length, Field Type X'10'
- Character String, Fixed Length, Nullable, Field Type X'90'
- Character String, Variable Length, Field Type X'11'
  - Mode X'01'
- Character String, Variable Length, Nullable, Field Type X'91'
  - Mode X'01'
- COBOL/2 Zoned Decimal Fixed Point, Field Type X'35'
  - Mode X'00'
- COBOL/2 Zoned Decimal Fixed Point, Nullable, Field Type X'B5'
  - Mode X'00'
- Decimal Fixed Point, Field Type X'30'
  - Mode X'00'
  - Scale between 0 and Field length
- Decimal Fixed Point, Nullable, Field Type X'B0'
  - Mode X'00'
  - Scale between 0 and Field length
- Fixed Point Numeric Character String, Field Type X'32'
  - Character size: 1 byte
  - Mode X'00'
- Fixed Point Numeric Character String, Nullable, Field Type X'B2'
  - Character size: 1 byte
  - Mode X'00'

- Hexadecimal Floating Point, Field Type X'40'
- Hexadecimal Floating Point, Nullable, Field Type X'C0'
- Null-Terminated Byte String, Field Type X'03'
  - Mode X'01'
- Null-Terminated Byte String, Nullable, Field Type X'83'
  - Mode X'01'
- Null-Terminated Character String, Field Type X'14'
- Null-Terminated Character String, Nullable, Field Type X'94'
- PC(8087) Signed Binary Integer, Field Type X'24'
  - 1, 2, or 4 byte length
- PC(8087) Signed Binary Integer, Nullable, Field Type X'A4'
  - 1, 2, or 4 byte length
- Short Byte string, Field Type X'07'
  - Mode X'01'
- Short Byte string, Nullable, Field Type X'87'
  - Mode X'01'
- Short Character String, Variable Length, Field Type X'19'
  - Mode X'01'
- Short Character String, Variable Length, Nullable, Field Type X'99'
  - Mode X'01'
- Signed Binary Fixed Point, Field Type X'31'
- Signed Binary Fixed Point, Nullable, Field Type X'B1'
- Signed Binary Integer, Field Type X'23'
  - 1, 2, or 4 byte length
- Signed Binary Integer, Nullable, Field Type X'A3'
  - 1, 2, or 4 byte length
- Unsigned Binary Fixed Point, Field Type X'34'
- Unsigned Binary Fixed Point, Nullable, Field Type X'B4'
- Unsigned Binary Integer, Field Type X'22'
  - 1, 2, or 4 byte length
- Unsigned Binary Integer, Nullable, Field Type X'A2'
  - 1, 2, or 4 byte length
- Zoned Decimal Fixed Point, Field Type X'33'
  - Mode X'00'

            — Zoned Decimal Fixed Point, Nullable, Field Type X'B3'
              — Mode X'00'
  — Auxiliary Triplets:
    — Implementation Support Data (ISD)
      — Subset X'0000'
    — Continue Preceding Triplet (CPT)

- FD:OCA Subset 0100, DRDA Support Tower
  — All constructs, parameters, and parameter values of the base subset, plus:
  — Attribute Triplets:
    — Metadata Definition (MDD)
      — Application Class 5
    — Simple Data Array (SDA)
      — Field Types:
        — VAX Binary Floating Point, Field Type X'49'
        — VAX Binary Floating Point, Nullable, Field Type X'C9'
  — Auxiliary Triplets:
    — Implementation Support Data (ISD)
      — Subset X'0100'

5.3 FD:OCA Compliance Rules

This section defines the compliance rules that apply for the generator and the receiver of an FD:OCA object.

5.3.1 Compliance Rules for the FD:OCA Object Generator

The generator of a Formatted Data Object may elect any of the functional subsets defined above, and must then describe the object with the facilities available in that subset.

The generator is said to comply with an FD:OCA subset if and only if all the used constructs and their parameter values are from the elected subset and none of their syntax rules are violated. The syntax rules for all constructs and their parameters are spelled out in Chapter 4 on page 25.

5.3.2 Compliance Rules for the FD:OCA Object Receiver

The object receiver may or may not be capable of handling a particular FD:OCA tower, or all the functions of a particular subset level in a tower. An object found to contain constructs and/or parameter values beyond the functional capabilities of the receiving product may just be ignored by the receiving product.

If the embedding environment or architecture explicitly says what tower and subset level the object uses, and if this conflicts with the tower and subset level supported by the receiver, then the receiver may ignore the object without inspecting it.

The same is true if the object is part of a group of objects, such as all those carried in a particular document, and the embedding environment or architecture indicates the FD:OCA tower and the maximum subset level required for the group. If this conflicts with the receiver's functional capabilities, then all objects in the group may be ignored without inspection.

If, however, the object is known to use only such parameter values and constructs that belong to the FD:OCA tower and subset level supported by the receiver, then the receiver must correctly interpret the object and understand its semantics, as defined in Chapter 4 on page 25.

5.4 Code Point Assignments

The table shown in Table 5-1 provides an overview of all FD:OCA triplets and their associated triplet type identifiers.

Table 5-1 FD:OCA Code Point Assignments

70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
SDA	RLO				GDA	GDA		MDD						ISD	CPT

The triplet type identifier is represented by its hexadecimal value shown in the first row of the table. The second row contains the associated triplet acronyms in the corresponding columns.

Glossary

Some of the terms and definitions that appear in this glossary have been taken from other source documents. Definitions reprinted from the *American National Dictionary for Information Processing Systems* are identified by the symbol (ANDIPS) following the definition. Definitions reprinted from working documents, draft proposals, or draft international standards of ISO Technical Committee 97, Subcommittee 1 (Vocabulary) are identified by the symbol (TC97) following the definition. Definitions reprinted from a published section of the ISO *Vocabulary-Information Processing* or from a published section of the ISO *Vocabulary-Office Machines* are identified by the symbol (ISO) following the definition.

If a term is not included here, refer to the *Dictionary of Computing* (SC20-1699, IBM).

The following definitions are provided as supporting information only, and are not intended to be used as a substitute for the semantics described in the body of this reference.

ANSI
: American National Standards Institute. An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. It is the United States constituent body of the International Standards Organization (ISO).

application
: The use to which an information system is put.

application program
: A program written for or by a user that applies to the user's work.

array
: The conceptual model used to describe formatted data. An array describes a string of data fields in terms of dimensions. See also dimension.

attribute
: A property or characteristic of one or more entities.

attribute triplets
: The part of a descriptor that defines the structure and representation of the data fields. (TC97)

base-and-towers concept
: A conceptual illustration of an architecture which shows the architecture as a base with optional tower(s). The base and the towers represent different degrees of function achieved by the architecture.

base support level
: Within the base-and-towers concept, the lowest permissible degree of function achieved by an architecture. This is represented by a base with no towers. Synonymous with mandatory support level.

BITS
: A data type for architecture syntax, indicating one or more bytes to be interpreted as bit string information.

CCSID
: Coded Character Set Identifier. A 16-bit number identifying a specific set of encoding scheme identifier, character set identifier(s), code page identifier(s), and other relevant

information that uniquely identifies the coded graphic character representation used.

CDRA
Character Data Representation Architecture. An IBM architecture that defines a set of identifiers, services, and conventions to achieve a consistent representation, processing, and interchange of graphic character data.

CGCSGID
Coded Graphic Character Set Global Identifier. A four-byte binary or a ten-digit decimal identifier consisting of the concatenation of a GCSGID and a CPGID. It identifies the code point assignments in a code page of a specific graphic character set, from among all the graphic characters that may be assigned in a code page.

CHAR
A data type for architecture syntax, indicating one or more bytes to be interpreted as character information.

character
A member of a set of elements used for the organization, control, or representation of data. A character can be a graphic character or a control character. (ISO) See also graphic character.

character code
An element of a code page or a site in a code table to which a character can be assigned. The element is associated with a binary value. The assignment of a character to an element of a code page determines the binary value that will be used to represent each occurrence of the character in a character string.

character identifier
The unique name for a graphic character.

character set
A finite set of different graphic or control characters that is complete for a given purpose; for example, the character set in ISO Standard 646, *7-bit Coded Character Set for Information Processing Interchange*.

character string
A sequence of characters.

CODE
A data type for architecture syntax, indicating an architected constant to be interpreted as defined by the architecture.

coded graphic character
A graphic character that has been assigned one or more code points within a code page.

coded graphic character set
A set of graphic characters with their assigned code points.

code page
A set of assignments, each of which assigns a code point to a character. Each code page has a unique name or identifier. Within a given code page, a code point is assigned to one character. More than one character set can be assigned code points from the same code page. See also code point.

code point
A unique bit pattern that can serve as an element of a code page or a site in a code table, to which a character can be assigned. The element is associated with a binary value. The assignment of a character to an element of a code page determines the binary value that will

be used to represent each occurrence of the character in a character string. Code points are either one byte or two bytes in length.

column
A sub-array consisting of all elements that have an identical position within the low dimension of a regular two-dimensional array.

CPGID
Code Page Global Identifier. A unique code page identifier that can be expressed as either a two-byte binary or a five-digit decimal value.

database
A collection of data fundamental to a system. An organized collection of user information that can be methodically created, updated, or retrieved. The database organization is usually defined and tailored to meet the specific needs of the user.

data element
A unit of data that is considered indivisible in a particular environment.

data stream
A continuous stream of data that has a defined format. An example of a defined format is a structured field.

data type
A classification of data into different types having some bearing on proper handling of that data. Floating point, Integer, Binary, and Character string are some examples.

DBCS
Double-Byte Character Set. A character set, such as a set of Japanese ideographs, that requires two bytes to identify each character.

default
An alternate value, attribute, or option that is assumed when none has been specified and one is needed to continue processing. An example of defaults is default drawing attributes.

dimension
Each successive level of partitioning. Defining dimensions allows the addressing of specific parts of an array. See also partitioning and array.

document
A machine-readable collection of one or more objects which represent a composition, a work, or a collection of data.

A publication or other written material.

document component
A set of related structured fields which are bounded by begin and end structured fields. Examples are object, page, and overlay.

document content architecture
A family of architectures that define the syntax and semantics of the document components that are allowed to appear in document content architecture data streams. See also document component and structured field.

document editing
A method used to create or modify a document.

document element
A self-identifying, variable-length, bounded record, which may have a content portion that provides control information, data, or both. An application or device does not have to

understand control information or data to parse a data stream when all of the records in the data stream are document elements. Synonymous with structured field.

DRDA
Distributed Relational Database Architecture. A protocol that allows applications to access data from remote databases.

EBCDIC
Extended Binary-Coded Decimal Interchange Code. A coded character set consisting of eight-bit coded characters.

element
A bar or space in a bar code character or a bar code symbol.

A structured field in a document content architecture data stream.

In FD:OCA, each of the data fields in an array.

A basic member of a mathematical or logical class or set.

Encoding Scheme
A set of specific definitions that describe the philosophy used to represent character data. The number of bits, the number of bytes, the allowable ranges of bytes, the maximum number of characters, and the meanings assigned to some generic and specific bit patterns are some examples of specifications to be found in such a definition.

ESID
Encoding Scheme Identifier. A number assigned to uniquely identify a particular encoding scheme specification.

exception
One of the following:

1. An invalid or unsupported data-stream construct.
2. In IPDS, a condition requiring host notification.
3. In IPDS, a condition that requires the host to resend data.

exception action
Action taken when an exception is detected.

exception condition
The condition that exists when a product encounters an invalid or unsupported construct.

extent
One of the characteristics of a dimension. If all partitions of a dimension have the same number of sub-partitions, then this number is called the extent of the next lower dimension. See also local extent.

factoring
The movement of a parameter value from one state to a higher-level state. This permits the parameter value to apply to all of the lower-level states unless specifically overridden at the lower level.

FDO
See formatted data object.

FD:OCA
Formatted Data Object Content Architecture. An architected collection of constructs used to interchange formatted data.

formatted data
Data whose implied syntax and semantics are represented by architected controls that accompany the data.

formatted data object (FDO)
An object that contains formatted data. See also object.

function set
A collection of architecture constructs and associated values. Function sets may be defined across or within subsets.

GCID
Another name for CGCSGID. See CGCSGID.

GCGID
Graphic Character Global Identifier. An eight-byte alphanumeric character string, used to identify a specific graphic character. It is from four to eight bytes in length.

GCSGID
Graphic Character Set Global Identifier. A unique graphic character set identifier that can be expressed as either a two-byte binary or a five-digit decimal value.

GID
See global identifier.

Global Identifier (GID)
One of the following:

- A Code Page Global ID (CPGID)
- A Graphic Character Global Identifier (GCGID)
- A Font Global Identifier (FGID)
- A Graphic Character Set Global Identifier (GCSGID)

In MO:DCA, an encoded graphic character string which, when qualified by the associated CGCSGID, provides a reference name for a document element.

graphic character
A member of a set of symbols which represent data. Graphic characters may be letters, digits, punctuation marks, or other symbols. See also character.

graphic character identifier
The unique name for a graphic character in a font, or in a graphic character set. See also character identifier.

hexadecimal
A number system with a base of sixteen. The decimal digits 0 through 9 and characters A through F are used to represent hexadecimal digits. The hexadecimal digits A through F correspond to the decimal numbers 10 through 15, respectively. An example of a hexadecimal number is X'1B', which is equal to the decimal number 27.

ID
Identifier.

IEC
International Electrotechnical Commission; an international standards writing body.

IEEE
Institute of Electrical and Electronics Engineers; a US standards writing body.

IPDS
Intelligent Printer Data Stream. An architected host-to-printer data stream that contains both data and controls defining how the data is to be presented.

ISO
International Standards Organization; an international standards writing body.

ISO/IEC
The prefix used for standards that are produced jointly by ISO and IEC.

Kanji
A graphic character set for symbols used in Japanese ideographic alphabets.

LID
See local identifier.

local extent
The number of sub-partitions within any given partition.

local identifier (LID)
An identifier that is mapped by the environment to a named resource.

location
A site within a data stream. A location is specified in terms of an offset in the number of structured fields from the beginning of a data stream, or in the number of bytes from another location within the data stream.

lowercase
Pertaining to small letters as distinguished from capital letters, Examples of small letters are a, b, and g. Contrast with uppercase.

mandatory support level
Within the base-and-towers concept, the lowest permissible degree of function achieved by an architecture. This is represented by a base with no towers. Synonymous with base support level.

Meaning
A table heading for architecture syntax. The entries under this heading convey the meaning or purpose of a construct. They may be long names, descriptions, or brief statements of function.

MO:DCA
Mixed Object Document Content Architecture. An architected, device-independent data stream for interchanging documents.

Name
A table heading for architecture syntax. The entries under this heading are short names that give a general indication of the contents of the construct.

nibble
A bit-pattern consisting of four bit.

object
A collection of structured fields. The first structured field provides a begin-object function and the last structured field provides an end-object function. The object may contain one or more other structured fields whose content consists of one or more data elements of a particular data type. An object may be assigned a name, which may be used to reference the object. Examples of objects are text, font, graphics, image, and formatted data objects.

object data
A collection of related data elements that have been bundled together. Examples of data elements are graphic characters, image data elements, and drawing orders.

Offset
A table heading for architecture syntax. The entries under this heading indicate the numeric displacement into a construct. The offset is measured in bytes and starts with byte zero. Individual bits may be expressed as displacements within bytes.

parameter
A variable that is given a constant value for a specified application and which may denote the application. (TC97) (ANDIPS)

partition
A conceptual subdivision of a string of data fields. A partition can be further divided into sub-partitions. See also dimension.

partitioning
A method used to place parts of a control into two or more segments or structured fields. Partitioning may cause difficulties for a receiver if one of the segments or structured fields is not received or is received out of order.

In FD:OCA, a conceptual division of a string of data fields into substrings. Each substring is called a partition. See also partition.

plane
A two-dimensional sub-array consisting of all elements that have an identical position within a given dimension of a regular three-dimensional array.

pragmatics
The part of a construct's description that describes the usage of the construct.

PTOCA
Presentation Text Object Content Architecture. An architected collection of constructs used to interchange and present presentation text data.

Range
A table heading for architecture syntax. The entries under this heading give numeric ranges applicable to a construct. The ranges may be expressed in binary, decimal, or hexadecimal. The range may consist of a single value.

regular array
An array in which all partitions of any dimension have the same number of sub-partitions. The individual elements of a regular array may or may not have identical format and length. See also array.

repeating group
A group of parameter specifications that may be repeated.

resource
An object that is referenced by a data stream or by another object to provide data or information. Resource objects may be stored in libraries. For example, in MO:DCA they may be contained within a resource group in the data stream. In IPDS, resources are downloaded to and stored by a printer. Examples of resources are fonts, overlays, and page segments.

row
A sub-array consisting of all elements that have an identical position within the high dimension of a regular two-dimensional array.

SBCS
Single-Byte Character Set. A character set that requires one byte to identify each character.

SBIN
A data type for architecture syntax, indicating one or more bytes to be interpreted as a signed binary number, with the sign bit in the high-order position of the leftmost byte.

semantics
The part of a construct's description that describes the function of the construct.

slice
A sub-array consisting of all elements that have an identical position within any given dimension of a regular n-dimensional array.

structured field
A self-identifying, variable-length, bounded record, which may have a content portion that provides control information, data, or both. Synonymous with document element.

structured field introducer
In MO:DCA, the header component of a structured field which provides information that is common for all structured fields. Examples of information that is common for all structured fields are length, function type, and set type. Examples of structured field function types are begin, end, data, and descriptor. Examples of structured field set types are presentation text, image, graphics, and page.

subset
Within the base-and-towers concept, a portion of architecture represented by a particular level in a tower or by a base. See also subsetting tower.

subsetting tower
Within the base-and-towers concept, a tower representing an aspect of function achieved by an architecture. A tower is independent of any other towers. A tower may be subdivided into subsets. A subset contains all the function of any subsets below it in the tower. See also subset.

syntax
The part of a construct's description that describes the structure of the construct.

tag
In FD:OCA, a special attribute triplet that can be attached to attribute triplets to provide them with additional information. In DRDA for example, an FD:OCA Metadata Definition triplet can express that a particular character field is actually a timestamp.

triplet
A three-part self-defining variable-length parameter consisting of a length byte, an identifier byte, and one or more parameter-value bytes. An example of the use of triplets is in a PTOCA Presentation Text Descriptor structured field to identify initial text conditions for modal control sequences.

triplet identifier
A one-byte type identifier for a triplet.

Type
A table heading for architecture syntax. The entries under this heading indicate the types of data present in a construct. The data type will be one of the following: BITS, CHAR, CODE, SBIN, UBIN, UNDF. See also these terms.

UBIN
A data type for architecture syntax, indicating one or more bytes to be interpreted as an unsigned binary number.

UNDF
A data type for architecture syntax, indicating one or more bytes that are undefined by the architecture.

uppercase
Pertaining to capital letters. Examples of capital letters are A, B, and C. Contrast with lowercase.

Index