

**Implementation of
ISO/IEC 11730 : 1994**

**Information technology —
Programming languages —
Form Interface Management
System (FIMS)**

Committees responsible for this British Standard

The preparation of this British Standard was entrusted by the Information Systems Technology Assembly (IST/-) to Technical Committee IST/5, upon which the following bodies were represented:

- British Computer Society
- Electricity Association
- Hoskyns Group plc
- IBM United Kingdom Ltd.
- Information Systems Committee of the Universities Funding Council
- International Computers Limited
- National Computer Users' Forum
- National Computer Centre Ltd.
- Rutherford Appleton Laboratory
- SEMA Group Systems Ltd.

This British Standard, having been prepared under the direction of the Information Systems Technology Assembly, was published under the authority of the Standards Board and comes into effect on 15 April 1995

© BSI 1995

The following BSI references relate to the work on this standard:
Committee reference IST/5
Draft for comment 93/640640DC

ISBN 0 580 23928 4

Amendments issued since publication

Amd. No.	Date	Text affected

National foreword

This British Standard reproduces verbatim ISO/IEC 11730 : 1994 and implements it as the UK national standard.

This British Standard is published under the direction of the Information Systems Technology Assembly whose Technical Committee IST/5 has the responsibility to:

- aid enquirers to understand the text;
- present to the responsible international committee any enquiries on interpretation, or proposals for change, and keep UK interests informed;
- monitor related international and European developments and promulgate them in the UK.

NOTE. International and European Standards, as well as overseas standards, are available from Customer Services, BSI, 389 Chiswick High Road, London WA 4AL.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

INTERNATIONAL STANDARD

ISO/IEC 11730

First edition
1994-12-01

Information technology — Programming languages — Form Interface Management System (FIMS)

*Technologies de l'information — Langages de programmation —
Système de gestion d'interface de forme (FIMS)*



Reference number
ISO/IEC 11730:1994(E)

AVA Documents
Document Sharing Hub

CONTENTS

Foreword	viii	Section 2: Conformance	
Introduction	ix	4 Conformance	21
Section 1: Overview		4.1 Categories	21
1 General	3	4.2 Conformance	21
1.1 Scope	3	4.2.1 Basic subset/full	21
1.2 Normative references	3	4.2.2 Synchronous/asynchronous	21
2 Definitions	5	4.3 Profile	22
3 FIMS philosophy of use	11	4.3.1 Device types	22
3.1 Design principles for the Full FIMS	11	4.3.2 Panning	22
3.2 Design principles for Basic Subset FIMS	12	4.3.3 Literal types	22
3.3 FIMS model	12	4.3.4 Presentation attributes	22
3.3.1 The terminal	13	4.3.5 Culture	22
3.3.2 The form control system (FCS)	15	4.3.6 Data input	22
3.3.3 The application program interface	15	4.3.7 Locator input	23
3.3.4 The form	15	4.3.8 Implementor names	23
3.3.5 The application program	16	4.3.9 Style elements	23
3.3.6 The operating environment	16	4.3.10 External panels	23
3.4 Relationships among display, form, and program	16	Section 3: Application interface	
3.5 Use of FIMS	17	5 Generic interface	27
		5.1 Introduction	27
		5.2 Request descriptions	27
		5.3 Cancel request	28
		5.4 Disable request	29
		5.5 Enable request	30
		5.6 Poll request	31
		5.7 Receive request	32
		5.8 Send request	33
		5.9 Transceive request	34
		5.10 Request parameters	35
		6 Language bindings	43
		6.1 Binding philosophy	43
		6.2 C binding	43
		6.2.1 FIMS subroutine names and calling sequences	43
		6.2.2 Optional and required parameters	44
		6.2.3 Structure definitions	44
		6.2.4 C datatypes for FIMS parameters	45
		6.2.5 Return values	46
		6.2.6 FIMS C binding library file	47

© ISO/IEC 1994

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

Section 4: Run-time processing

7	FCS processing model	53
7.1	Purpose	53
7.2	External events	53
7.2.1	Cancel request	54
7.2.2	Disable request	54
7.2.3	Enable request	54
7.2.4	Main program request	55
7.2.5	Poll request	55
7.2.6	Receive request	56
7.2.7	Send request	57
7.2.8	Transceive request	59
7.2.9	Send control text	60
7.3	Activation lists and accept phase for Basic Subset FIMS	60
7.3.1	First activation item for Basic Subset FIMS	60
7.3.2	Basic subset processing model	60
7.3.3	Functions and receive control text	61
7.4	Activation lists for Full FIMS	61
7.5	Accept phase and responses for Full FIMS	62
7.5.1	Determination of the first activation item	62
7.5.2	The current and next activation items	63
7.5.3	Philosophy of accept phase response processing	63
7.5.4	Entry, exit, and validation checks	64
7.5.5	Integration of locator input with accept phase	65
7.5.6	Menu processing	66
7.5.7	Processing models	68
7.6	Form data validation	79
7.7	Modified fields	80
7.8	Asynchronous requests	81
7.9	Request synchronization	82
7.9.1	Synchronization not required	82
7.9.2	Synchronization required	82
7.9.3	Synchronization rules	83
7.10	Display management	84
7.10.1	Viewport/window management	84
7.10.2	First display of a panel	85
7.10.3	Stacking order	85
7.10.4	Changed data items	86
7.11	Returning from the FCS	87
7.11.1	Status return	87
7.11.2	Unrecoverable exceptions	87
7.11.3	Timeout exception	88
7.12	Protection model	88

Section 5: IFDL specification

8	IFDL elements	93
8.1	Language definition method	93
8.1.1	Syntax diagram	93
8.1.2	Syntax rule	95
8.1.3	Semantic rule	95
8.2	Character set	95
8.2.1	Source character sets	95
8.2.2	Basic character set	95
8.2.3	Extended character set	96
8.2.4	Execution character sets	96
8.3	Language structure	96
8.3.1	Separators	96
8.3.2	IFDL-character-strings	97
8.3.3	IFDL picture-strings	98
8.3.4	Reference format	98
8.3.5	Form modularization	99
8.4	Constant-name	100
8.5	Constants	101
8.5.1	Constant	101
8.5.2	Date-time string	102
8.5.3	Logical constant	105
8.5.4	Number	106
8.5.5	String	109
8.6	Scope of definition and reference	110
8.7	User name qualified references	110
8.8	Data references	112
8.8.1	Data reference	112
8.8.2	Scalar-data reference	113
8.8.3	Array-data reference	114
8.8.4	Corresponding-data reference	115
8.9	Data structure references	117
8.9.1	Scalar-data-structure reference	117
8.9.2	Array-data-structure reference	118
8.9.3	Data-structure-selector	119
8.9.4	Data-structure-range	120
8.10	Field references	121
8.10.1	Scalar-field reference	121
8.10.2	Array-field reference	123
8.11	Group references	124
8.11.1	Scalar-group reference	124
8.11.2	Group-nesting	125
8.11.3	Array-group reference	126
8.11.4	Group-range-nesting	127
8.11.5	Group-selector	128
8.11.6	Group-range	129
8.12	Button references	130
8.12.1	Scalar-button reference	130
8.12.2	Array-button reference	131
8.13	Panel references	132

ISO/IEC 11730:1994(E)

8.14	Viewport references	133	9.8	Attribute declaration	185
8.15	Menu references	134	9.9	Barcode field declaration	186
8.16	Packet-field reference	135	9.10	Bar menu declaration	188
8.17	Subscripts	136	9.11	Bitmap literal declaration	189
8.17.1	Singular-subscript	136	9.12	Builtin function clause	190
8.17.2	Range-subscript	137	9.13	Bundle declaration	191
8.18	Data classes and data types	138	9.14	Button properties	193
8.19	Data assignment	139	9.15	Calculate clause	194
8.19.1	Character class assignment	140	9.16	Call clause	195
8.19.2	Integer class assignment	140	9.17	Call step	196
8.19.3	Decimal class assignment	140	9.18	Cascade button declaration	197
8.19.4	Float class assignment	141	9.19	Case step	198
8.19.5	Logical class assignment	141	9.20	Color clause	199
8.19.6	Date-time class assignment	141	9.21	Concealed clause	200
8.20	Data typing of special objects	141	9.22	Constant declaration	201
8.20.1	FIELD_VALUE data item	141	9.23	Control text response declaration	202
8.20.2	The fn%de_edit function	143	9.24	Copy statement	203
8.21	Expressions	144	9.25	Culture declaration	205
8.21.1	Expression	144	9.26	Current highlight clause	206
8.21.2	Scalar-expression	145	9.27	Data-type clause	207
8.21.3	Corresponding-expression	145	9.28	Data validation clause	210
8.21.4	Corresponding expression processing	145	9.29	Deactivate step	213
8.22	Character expressions	146	9.30	Default clause	214
8.22.1	Scalar-character-expression	146	9.31	Delete step	217
8.22.2	Corr-character-expression	146	9.32	Deselect step	218
8.22.3	Character-expression	147	9.33	Device declaration	219
8.22.4	Character-function	148	9.34	Disable response declaration	221
8.23	Date-time expressions	150	9.35	Display attribute	222
8.23.1	Scalar-date-time-expression	150	9.36	Display clause	223
8.23.2	Corr-date-time-expression	150	9.37	Display group clause	225
8.23.3	Date-time-expression	151	9.38	Display item clause	226
8.24	Logical expressions	154	9.39	Display menu clause	227
8.24.1	Scalar-logical-expression	154	9.40	Display panel clause	228
8.24.2	Scalar-when-expression	154	9.41	Display step	229
8.24.3	Corr-when-expression	154	9.42	Display viewport clause	231
8.24.4	Logical-expression	155	9.43	Editing clause	232
8.24.5	Accept-phase-condition	156	9.44	Enable response declaration	234
8.24.6	Relation	160	9.45	Enter step	235
8.24.7	Logical-function	162	9.46	Entry response declaration	237
8.25	Numeric expressions	163	9.47	Exit response declaration	238
8.25.1	Scalar-numeric-expression	163	9.48	Exit step	239
8.25.2	Corr-numeric-expression	163	9.49	Extent clauses	240
8.25.3	Numeric-expression	164	9.50	External panel declaration	241
8.25.4	Numeric-function	166	9.51	External response declaration	243
9	IFDL Statements	169	9.52	Field declaration	244
9.1	IFDL structure	169	9.53	Field general properties	245
9.2	Accept response declaration	171	9.54	Field text properties	247
9.3	Activate step	172	9.55	Formatted group declaration	251
9.4	Arc literal declaration	175	9.56	Formatted panel declaration	253
9.5	Area attribute	176	9.57	Form data declaration	254
9.6	Array displays clause	179	9.58	Form declaration	259
9.7	Array properties	182	9.59	Function declaration	261
			9.60	Function response declaration	265
			9.61	General attribute	271

9.62	Group declaration	272	9.116	Refresh step	367
9.63	Group properties	273	9.117	Remove step	368
9.64	Highlight clause	274	9.118	Reset step	369
9.65	If step	276	9.119	Response step clause	370
9.66	Import literal declaration	277	9.120	Return step	372
9.67	Input picture clause	279	9.121	Scrollbar clause	374
9.68	Insert step	283	9.122	Scroll clause	375
9.69	Internal response declaration	284	9.123	Secondary stream declaration	377
9.70	Invalid step	285	9.124	Select step	378
9.71	Invoke step	286	9.125	Send response declaration	379
9.72	Item attribute	287	9.126	Separator literal declaration	380
9.73	Label clause	288	9.127	Signal step	381
9.74	Language declaration	289	9.128	Simple group declaration	382
9.75	Layout declaration	291	9.129	Slider field declaration	383
9.76	Let step	295	9.130	Table declaration	385
9.77	Line attribute	296	9.131	Text attribute	387
9.78	List declaration	299	9.132	Text field declaration	390
9.79	Literal declaration	300	9.133	Text literal declaration	393
9.80	Literal properties	301	9.134	Title clause	396
9.81	Local clause	302	9.135	Tiled panel declaration	398
9.82	Location clauses	303	9.136	Timeout clause	399
9.83	Locator cursor clause	305	9.137	Toggle field declaration	400
9.84	Loop step	306	9.138	Transceive response declaration	402
9.85	Main program response declaration	308	9.139	Transient menu declaration	403
9.86	Margins clause	309	9.140	Type definition declaration	405
9.87	Menu declaration	310	9.141	Use help message clause	408
9.88	Menu properties	311	9.142	Use help panel clause	409
9.89	Message clause	312	9.143	Use menu clause	410
9.90	Message panel declaration	313	9.144	Validate step	411
9.91	Message step	315	9.145	Validation response declaration	414
9.92	Move step	316	9.146	Value clause	415
9.93	Option field declaration	317	9.147	Viewport attribute	416
9.94	Orientation clause	318	9.148	Viewport declaration	418
9.95	Output picture clause	319			
9.96	Packet declaration	324			
9.97	Packet identifier clause	331			
9.98	Packettlist declaration	332			
9.99	Panel declaration	333			
9.100	Panel properties	334			
9.101	Pattern clause	336			
9.102	Picture character clause	337			
9.103	Picture field declaration	338			
9.104	Picture string	340			
9.105	Polyline literal declaration	347			
9.106	Position step	348			
9.107	Print step	355			
9.108	Prompt clause	357			
9.109	Protected clause	358			
9.110	Push button declaration	360			
9.111	Radio toggles clause	362			
9.112	Rastercolor declaration	363			
9.113	Raster declaration	364			
9.114	Receive response declaration	365			
9.115	Rectangle literal declaration	366			

Section 6: FIMS device profiles

10	FIMS device profiles	423
10.1	The concept of a device profile	423
10.2	Pixel minimum	423
10.3	Character field	424
10.4	Character blockmode	425
10.5	Serial	426

Annex A: IFDL Philosophy of Use

A.1	Form concepts	427
A.2	Organization of the form	428
A.3	Form data	429
A.4	Packets	430
A.4.1	Packet / data associations	431
A.4.2	Altering name associations	431
A.4.3	Variant packets	432
A.4.4	Packet lists	433

ISO/IEC 11730:1994(E)

A.5	Shadow packets	433	A.16.7	Poll actions	468
A.5.1	Receive shadow packet	433	A.16.8	Cancel actions	468
A.5.2	Send shadow packet	435	A.16.9	The activation list	468
A.5.3	Use of shadow packets for navigation	436	A.17	Accept phase	468
A.6	Layouts	437	A.18	FCS responses	469
A.6.1	The display	437	A.18.1	Types of responses	469
A.6.2	Display objects	438	A.18.2	Scope of responses	474
A.6.3	IFDL style objects and properties	441	A.18.3	Response execution	474
A.6.4	Viewports	441	A.18.4	Response steps	475
A.6.5	Panels	441	A.19	Functions	477
A.6.6	Relationship to windowing systems	442	A.19.1	Association of functions with input devices	477
A.6.7	Types and properties of panels	442	A.19.2	Examples of function handling	477
A.7	Display attributes	444	A.20	Locators	479
A.8	Literals	444	A.21	Sub-activation lists	479
A.9	Field declaration	445	A.22	Help model	480
A.9.1	Field session properties	445	A.22.1	Basic elements of the help model	480
A.9.2	Field appearance properties	446	A.22.2	Default help model	480
A.9.3	Field processing properties	446	A.22.3	Complex help processing	481
A.9.4	Field validation properties	447	A.23	FCS/operator interaction	481
A.9.5	Picture fields	448	A.23.1	Next item and previous item	482
A.9.6	Text fields	450	A.23.2	Next panel and previous panel	482
A.9.7	Toggle fields	451	A.23.3	Terminate normal	482
A.9.8	Option fields	451	A.23.4	Terminate immediate	482
A.9.9	Slider fields	452	A.23.5	Occurrence movement: up, down, left, and right	482
A.9.10	Barcode fields	452	A.23.6	Exit group next and previous	483
A.10	Protection of panel fields during operator input	452	A.23.7	Refresh device	483
A.11	Lists and tables	452	A.23.8	Focus Change	483
A.11.1	Lists	453	A.24	Printer and other secondary devices	483
A.11.2	Tables	453	A.24.1	Primary (batch) printing	483
A.12	Buttons	454	A.24.2	Secondary device output	484
A.13	Menus	454	A.25	Transaction processing	484
A.13.1	Menu display objects	454	A.26	Relationship with other device management systems	485
A.13.2	Bar menu	455			
A.13.3	Transient menu	455			
A.13.4	Selecting and activating menus	455			
A.13.5	Menu example	455	Annex B: Worked examples		487
A.14	Data composition	456	B.1	IFDL form	488
A.14.1	Structures	456	B.2	C program	492
A.14.2	Arrays	457			
A.14.3	Data type definition	466	Index		
A.15	Display object defaults	466			
A.16	FCS actions for external requests	467			
A.16.1	Main program actions	467			
A.16.2	Enable actions	467			
A.16.3	Disable actions	467			
A.16.4	Send actions	467			
A.16.5	Receive actions	468			
A.16.6	Transceive actions	468			

Figures

1	FIMS Model	13	18	Logical Picture Characters	346
2	Device Terminology, Simple Case	13	19	Display Objects	438
3	Multiple Displays on a Display Device	14	20	Windowing Systems / FIMS Correspondence	442
4	Viewports on a Display	14	21	Effect of sign, currency, and replacement characters in pictures	449
5	Simple and Complex Sizes	14	22	Limit cases for repeating and floating replacement	450
6	The Two Interfaces in FIMS	16	23	Date picture examples	450
7	A Possible Implementation of FIMS	18	24	Scrolling Possibilities with the POSITION Response Step	464
8	Three Dimensions Displayed on Two Dimensions	180	25	Types of Events	469
9	Suggested text field labelling	392	26	Legal Placement of Responses for Full FIMS	470
10	Orientation and language path	395			
11	Views of a session	427			
12	Hierarchical Form Organization for Full FIMS	428			
13	Hierarchical Form Organization for Basic Subset FIMS	429			
14	Display Coordinate System	438			
15	Windowing System Model	442			
16	Text Field with LANGUAGE PATH RIGHT	451			
17	Sizing a Formatted Group	462			
18	Effect of Vertical Scrolling	463			
19	Effect of Horizontal Scrolling	463			

Tables

1	Request Parameters	35
2	Control Texts returned from the FCS	40
3	Short Request Names for C	43
4	FIMS Parameters and C Datatypes	45
5	C Return Status	47
6	IFDL Character Set	96
7	Data Classes	138
8	Data Conversion Functions	139
9	Converting Form Data to Pictures	139
10	Relational Operators	160
11	Barcode types and strings	186
12	Legal locations for display and default clauses	223
13	Sign Output	233
14	Default Function Responses	268
15	Standard Paper Sizes	293
16	Character, Numeric, and E-notation Picture Characters	344
17	Date-time Picture Characters	345

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 11730 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Annexes A and B of this International Standard are for information only.

Introduction

In 1979, the CODASYL COBOL Committee (CCC) established an *ad hoc* group, the Screen Management Task Group (SMTG), whose objective was to provide the COBOL programmer with language facilities for manipulating two-dimensional visual display units. In order to meet overall COBOL objectives for portable programs and to provide meaningful capabilities for terminals hooked directly to processors, for terminals that reach "their" processor over a network, and for micro-computers which can act in the role either of "processor" or of "terminal", a very flexible facility was needed.

The CCC recognized that such a facility was of benefit not only to the COBOL community, but to the data processing community at large. The SMTG's charter was enlarged to include a language independent Screen Management System, with a COBOL Interface. The SMTG made its final report to the CCC in 1984 with a document that formed the base for the present standard.

In 1986, the CODASYL Committee was accredited by ANSI as a standards sponsor using the canvass method. The CODASYL Committee then formed the CODASYL Form Interface Management Committee to create, improve, and maintain this standard. Work on the standard began in 1987. The "Screen Management" name was changed shortly afterward to "Form Interface Management" so that it would reflect more accurately the scope of the work—it is an interface system to all sorts of devices and it uses an entity called a form to describe the interface.

In October, 1988, ISO/IEC JTC1 SC22 formed Working Group 18 for the purpose of producing an International Standard for FIMS.

In August, 1990, the two groups approved this document as a Draft CODASYL Standard with direction that it proceed to the ISO track for balloting. It was registered as an ISO CD (Committee Draft) in September, 1991 and was immediately followed by a successful ISO/IEC SC22 CD ballot and a simultaneous, successful, U.S.-only, CODASYL canvass, ending in February 1992.

In February, 1993 the Draft International Standard (DIS) version of the document started balloting in ISO and finished successfully in August, 1993. The ANSI Canvass started in May, 1993 and ended in September, 1993. Both were successful. The committees made the final changes to the document according to comments received in November, 1993. The final edits were completed in January, 1994. At that time the document became a CODASYL Standard (CS-1:1994) and ISO 11730:1994.

Only certain parts of this document are normative: Clauses 1 through 10. In those clauses, examples and footnotes are not part of the standard. The annexes are informative.

This International Standard is composed of five sections, as follows:

Section 1: Overview

Clause 1, General, describes the scope of the International Standard and lists other International Standards referenced by this International Standard. Clause 2, Definitions provides definitions for terms unique to FIMS. Clause 3, FIMS philosophy of use, describes the components of the Form Interface Management System. It shows how an application program communicates with a device acting as a terminal. When the application acts as the main program, it uses an interface (based on the Generic Interface) to invoke a form (described in IFDL). While not part of the International Standard, Annex A, IFDL Philosophy of Use, describes in general terms the *intent* of the IFDL. It gives an overview of the IFDL and the processing that will be performed on a form defined in IFDL. Likewise, Annex B, Worked examples, is useful in achieving an overview of how an application program and a form fit together.

Section 2: Conformance

The FIMS is a complex system, consisting of two major categories of two levels each of capabilities. Because of the wide variety of terminal capabilities, there are also a large number of implementor options that state what an implementation provides. The rules that govern whether an implementation conforms to the FIMS International Standard are stated in Clause 4, Conformance.

Section 3: Application interface

Clause 5, Generic interface, is an abstract definition of application programs can invoke the forms defined in IFDL. The generic interface is not a specification for a particular programming language. It does, however, specify what information shall be included in such a specification. Clause 6, Language bindings, describes the state of some language bindings.

Section 4: Run-time processing

The run-time component of the FIMS is the Form Control System (FCS). It implements a predefined processing model, easing the burden of user interface programming. Clause 7, FCS processing model, describes precisely how the FCS uses the IFDL to interact with the application program and the terminal.

Section 5: IFDL specification

The Independent Form Description Language is used to define a *form* which is the means of describing interactions with the terminal and the application program. Clause 8, IFDL elements, describes the metalanguage used to describe IFDL; it also contains the elements of IFDL descriptions: reference format, constants, name references, and expressions. Clause 9, IFDL Statements, contains the syntax and semantics of IFDL statements.

Section 6: FIMS device profiles

Several FIMS device profiles are described in order to enhance portability between implementations. Since FIMS allows implementations a major role in deciding what attributes of devices to support, portability would be a problem if all implementations were different. Clause 10, FIMS device profiles, describes specifics of common devices so that form designers can know what features are supported across implementations. A FIMS implementation is not limited to the features of these common devices; rather, these devices are a subset of the devices that FIMS implementation can use.

Information technology — Programming languages — Form Interface Management System (FIMS)

Section 1: Overview

Clause 1, General, describes the scope of this International Standard and lists other International Standard referenced by this International Standard. Clause 2, Definitions provides definitions for terms unique to FIMS. Clause 3, FIMS philosophy of use, describes the components of the Form Interface Management System. It shows how an application program communicates with a device acting as a terminal. When the application acts as the main program, it uses an interface

(based on the Generic Interface) to invoke a form (described in IFDL). While not part of this International Standard, Annex A, IFDL Philosophy of Use, describes in general terms the *intent* of the IFDL. It gives an overview of the IFDL and the processing that will be performed on a form defined in IFDL. Likewise, Annex B, Worked examples, is useful in achieving an overview of how an application program and a form fit together.

ISO/IEC 11730:1994(E)

1 General

1.1 Scope

This International Standard specifies a software system for description and control of human interfaces to computer input/output devices. It contains the following components.

- a) The Independent Form Description Language (IFDL), a language for the description of forms, which consist of declarations and procedural statements specifying the appearance and control of a dialog. FIMS has specific constructs for the appearance of forms on PCs, character devices, blockmode devices, and high quality printers.
- b) The Form Control System, the runtime component that implements the FIMS model for processing requests from programs and for interaction with a human operator. The FIMS model includes a dialog management system and data management discipline.
- c) A generic (or abstract) interface, specifying information that is passed between an application program and the FCS.
- d) A binding of the generic interface to the C language (ISO/IEC 9899).
- e) A profile of minimal, common devices, one of which shall be supported by an implementation.
- f) A set of conformance rules that describe choices for levels of FIMS and a set of implementor choices that shall be specified by each implementation to create a profile for that implementation.

This International Standard does not define a windowing system or a windowing system toolkit of GUI interface artifacts. Instead, it expects such systems to already be available and it builds on them, adding value to the usual toolkit implementations of such systems.

This International Standard does not define a "recommended user interface" or "style". Instead, it is defined to allow the style of the underlying windowing system to be expressed.

1.2 Normative references

The following International Standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All International Standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the International Standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 646:1991, *Information technology—ISO 7-bit coded character set for information interchange*

ISO/IEC 9899:1990, *Programming languages—C*

ISO 8601:1988, *Data elements and interchange formats - Information interchange—Representation of dates and times*

ISO/IEC TR 10182:1993, *Information technology—Programming languages, their environments and system software interfaces—Guidelines for language bindings*

1.2 Normative references

ISO/IEC 11730:1994(E)

2 Definitions

For the purposes of this International Standard, the following definitions apply.

- 2.1 Accept Phase:** That part of external event processing in which the FCS allows the operator to enter input for each activation item on the activation list.
- 2.2 Activation List:** An ordered list made up of activation items which causes input to be solicited from the operator during accept phase.
- 2.3 Activation Item:** An element of an activation list that specifies what input is allowed from the operator. In Full FIMS, an activation item is either a field (which allows the operator to enter data into that field) or a button (which allows the operator to position the cursor or locator to the field and enter a function). In Basic Subset FIMS, an activation item is the name of a panel, all of whose unprotected fields are available for input.
- 2.4 Active:** The state of a displayable item that is on the activation list and is not protected.
- 2.5 Active Item:** An activation item that has not been deactivated and is not protected.
- 2.6 Active Object:** See Active Item.
- 2.7 Active Request:** A request that is currently being processed for a particular session by the FCS. There is only one active request per session.
- 2.8 Address Space:** In an operating environment, the set of data accessible to a program.
- 2.9 Ancestor Object:** A display object that is a parent object or a parent object of an ancestor object.
- 2.10 Application Declarations:** A subset of a host language's data description syntax that describes the format of the messages that pass between the program and the form.
- 2.11 Application Program:** That part of an application that is concerned with data manipulation rather than operator interface.
- 2.12 Application Record:** A data area in the application program that contains a record that will be passed as a packet.
- 2.13 Array:** Multiple objects with the same name that can be distinguished by a subscript.
- 2.14 Asynchronous Request:** A request that does not have to complete before the FCS returns control to the calling program.
- 2.15 Attribute:** An appearance or presentation related possibility for display object that does not affect behavior or processing.
- 2.16 Basic Character Set:** Same as IFDL character set.
- 2.17 Button:** A display object that is capable of receiving only function input and that does not necessarily map a form data item.
- 2.18 Canceled Request:** A request that was pending or active when a CANCEL request specifying it was received by the FCS.
- 2.19 Character Data:** Data that has been declared with the character data-type.
- 2.20 Character Picture:** A picture that allows a subset of characters from a particular character set in its data positions.
- 2.21 Child Object:** A display object declared in another display object.
- 2.22 Completed Request:** A request that has finished its FCS processing.
- 2.23 Concealed:** A property of a field that directs that the value of the field is not displayed to the operator.
- 2.24 Control Message:** A message that passes between the program and the FCS. A control message is made up of 0 to 5 control text strings, each having five characters.
- 2.25 Control Text:** A string of five characters included in a control message. A control text string sent to a form can have any contents agreed on by the form designer and the programmer. A control text received by the program has a special format for its first two characters; it can contain status information from the FIMS on the results of form request statement as well as control text specified in the form to report on special conditions.

- 2.26 Corresponding Data:** A form data item in an array that identifies the occurrence in the same relative position of the array as the item that contains the reference. *(This item is not available in Basic Subset FIMS.)*
- 2.27 Corresponding Receive Field:** A field that does not have the unconditional PROTECTED property and that maps through its associated data item to a packet field in the current receive packet. *(This item is not available in Basic Subset FIMS.)*
- 2.28 Corresponding Send Field:** A field that has either the same name as a packet field in the current packet or a USING or DESTINATION clause referencing it in a packet field declaration for the current packet. Such a field identifies its containing panel as a target of the ACTIVATE CORRESPONDING SEND PANELS response step. *(This item is not available in Basic Subset FIMS.)*
- 2.29 Current Activation Item:** The activation item that is currently being processed in the accept phase model of processing. Equivalent to "focus". The item to which keyboard or other non-locator input is directed.
- 2.30 Current Packet:** The data area at runtime that contains the packet for the FCS operation in progress.
- 2.31 Current Value:** The value of a form data item as last set in the form. The current value will be used in the next reference to the form data item. Compare to LAST-KNOWN VALUE. *(This item is not available in Basic Subset FIMS.)*
- 2.32 Cursor:** A visual indicator on a video terminal that tells the operator where keyboard input will be directed. In FIMS, this corresponds to the field or button which is the current activation item.
- 2.33 Data Class:** A set of data types with similar properties such that data items declared with those types can take part in operations with each other without explicit conversion required in syntax.
- 2.34 Data Collection Phase:** That part of external event processing in which the FCS moves values from form data items into the packet fields of a packet.
- 2.35 Data Dictionary:** A tool used in the development of applications. A data dictionary is a centralized storehouse for names and attributes (such as data types, initial values, structure and validation information); names and attributes so stored can be accessed by multiple applications and parts of applications.
- 2.36 Data Distribution Phase:** That part of external event processing in which the FCS moves values from the packet fields of a packet into form data items.
- 2.37 Data Picture Character:** Characters in a picture string that represent a position for data to be displayed or be entered; does not refer to picture characters that invoke Insertion strings or editing of any type.
- 2.38 Data Structure:** A set of form data items that are related to each other by the structure name.
- 2.39 Data Type:** A complete set of properties describing the meaning and allowable operations on a data item.
- 2.40 Decoration:** The parts of a window in a windowing systems that are part of the windowing system rather than part of the application inside the window. For example, border, resize handles, operator menu, minimize and maximize buttons, title bar.
- 2.41 Descendent Object:** A display object that is a child object or a child of a child object.
- 2.42 Device:** An input or output mechanism.
- 2.43 Display:** The rectangular area mapped to the display device that is available to a particular form.
- 2.44 Display Attribute:** Terminal dependent video rendition characteristic applied to the image of display object.
- 2.45 Display Device:** The part of a terminal that presents output to an operator.
- 2.46 Display Object:** An IFDL entity that maps to a device for presentation or input purposes.
- 2.47 Environment:** The implementation defined facilities available to an application program and FCS.
- 2.48 Escape Unit:** A program invoked by the FCS in an implementor defined manner to service a CALL response step. The program so invoked may accept parameters, modify them, or call on the FCS directly. *(This item is not available in Basic Subset FIMS.)*
- 2.49 Event:** An occurrence that provokes the FCS to take some action.
- 2.50 Extended Character Set:** A character set supported by an implementation that contains characters other than the basic character set.

- 2.51 External Panel:** A panel whose contents and processing are specified by the application program through calls on a non-FIMS presentation system, such as GKS. (*This item is not available in Basic Subset FIMS.*)
- 2.52 External Event:** A request made by the application program to the FCS to perform some service.
- 2.53 FCS:** Form Control System.
- 2.54 Field:** A display object that maps an element of form data to the display within a panel.
- 2.55 Field Image:** The appearance of a field in a panel on the display. A field's image may be different on different panels. The field image consists of the field image value and any display attributes assigned to the value.
- 2.56 Field Image Value:** The characters or graphics making up a field image on the display; such characters and graphics are output to the display device to produce whatever effect it is capable of.
- 2.57 FIMS:** Form Interface Management System.
- 2.58 Focus:** See Current Activation Item.
- 2.59 Form:** A named, logical entity that is described in the IFDL, consisting of information to be loaded at run time and to be interpreted by the FCS. The form contains all the terminal-dependent information within the FIMS. It describes all the display interactions together with the data that is transferred and any processing that is to occur.
- 2.60 Formatted Group:** A type of group display object that maps a data structure to a panel and in which fields and literals have all their location and display properties specified by the form designer.
- 2.61 Formatted Panel:** A type of panel object in which the contained display objects have their location and display properties specified by the form designer.
- 2.62 Form Control System:** The run time part of the FIMS that handles the transfer and processing of data between the program and the terminal.
- 2.63 Form Data:** All variable data declared in the form.
- 2.64 Form Interface Management System (FIMS):** A software system that provides the capability to interact with input and output devices. It consists of the Independent Form description Language and the Form Control System.
- 2.65 Form Request Statement:** A statement in a host language that initiates activity in the FCS.
- 2.66 Function:** A subset of the input whose invocation is intended to cause immediate response rather than data entry. One example of a function is a "help" key on a keyboard.
- 2.67 Graphic Character:** A character that produces a visual image when sent to a terminal; a non-control character.
- 2.68 Group:** A class of display objects that provides a named scope and that contains fields, buttons, literals, and other groups.
- 2.69 Image Message:** Information that passes between the display and the form.
- 2.70 Implementation Default Character Set:** The character set that is supported by an implementation at run-time and that contains at least those characters from the basic character set.
- 2.71 Independent Form Description Language:** The language in which forms are described.
- 2.72 Input Device:** That part of a terminal through which data or control information is entered. The device most commonly used is a typewriter-style keyboard. Other examples are light-pen, touch panel, voice, and mouse.
- 2.73 Internal Response:** A sequence of response step that can be executed from another response. (*This item is not available in Basic Subset FIMS.*)
- 2.74 Invalid Status:** A property of a form data item stating that its value is invalid and the operator shall enter a new value before it is considered to be valid.
- 2.75 IFDL:** Independent Form Description Language.
- 2.76 IFDL Character Set:** The set of characters that make up the IFDL language. This set determines the legal user words, keywords, and punctuation of the language.
- 2.77 Last-Known Value:** The value of a form data item that is known by the program. A value is known in one of two ways: it is set when a program sends a packet which transfers data to the form data item, or it is set when the new value of a form data item is transferred to the program in a receive packet. (*This item is not available in Basic Subset FIMS.*)
- 2.78 Layout:** A terminal dependent view of the form, consisting of a set of panels and other properties.

- 2.79 Literal:** A display object (text string, line, point, arc) whose only purpose is to appear as an image on the display device.
- 2.80 Locator:** A generic name for input devices that are capable of pointing to a video display and that can deliver information (functions and presence) associated with particular display objects independent of the focus.
- 2.81 Main Program:** The first routine that gets control from the operating system when a run unit is started.
- 2.82 Menu:** A display object that allows selection of one or more items. *(This item is not available in Basic Subset FIMS.)*
- 2.83 Message Panel:** A panel used (by both the FCS and the application program) to display unformatted informational and error messages to the terminal operator.
- 2.84 Modified Field:** A packet field whose value is different from the value last known to the application for that field.
- 2.85 Numeric Form Data:** Form data that has been declared with one of the integer, decimal, or float data-types.
- 2.86 Numeric Picture:** A picture that has only 9's for data positions.
- 2.87 Operating Environment:** The set of services provided by an implementation to the FCS including, but not limited to, operating system and utility routines.
- 2.88 Operator:** The person viewing data on a display device and/or entering data through an input device at runtime.
- 2.89 Operator Control Menu:** In a windowing system, a menu that an operator can invoke to control windowing system functions. Such a menu is usually part of a viewport's decorations and is not under control of FIMS.
- 2.90 Packet:** A message that passes between the object program and the form, containing data or information about data.
- 2.91 Panel:** A named set of display objects.
- 2.92 Panel Unprotected List:** In the processing model for block mode terminals, all activation items that are on the same panel as the current activation item, that are contiguous on the activation list, and that are not protected.
- 2.93 Panning:** The action of an operating environment that allows objects in a panel to be viewed in a viewport that is too small to display all of them simultaneously.
- 2.94 Parent Object:** A display object containing another display object.
- 2.95 Parent Request:** A request that was previously active and that invoked a procedural escape that in turn makes a request to the FCS. *(This item is not available in Basic Subset FIMS.)*
- 2.96 Pending Request:** A request that is waiting to become the active request.
- 2.97 Popup Menu:** A menu that is associated with a panel and that is not displayed until it is requested by operator action in the panel.
- 2.98 Procedural Escape:** A call from the FCS to another program (called an escape unit) invoked in an implementor defined manner. *(This item is not available in Basic Subset FIMS.)*
- 2.99 Process:** The binding, in an operating environment, of a program to an address space.
- 2.100 Program:** See Application Program.
- 2.101 Prompt:** A message to an operator about what action is expected for the focus item.
- 2.102 Protected:** A property specifying that a field is not available for input.
- 2.103 Property:** A characteristic of an object.
- 2.104 Pulldown Menu:** A menu that is associated with a cascade button in a bar menu and that is not displayed until it is requested by operator action.
- 2.105 Receive Shadow Packet:** A message returned to the program containing indicators about each packet field in the associated receive packet; depending on the call, the indicators contain one of two types of information: 1) whether the packet fields have changed value from the values last known to the program or 2) location of the cursor at the conclusion of the last call. A receive shadow packet contains one character for each packet field in the receive packet plus one additional character.
- 2.106 Request ID:** A string containing information about the status of an asynchronous request.
- 2.107 Request Queue:** A place where pending requests are kept.

- 2.108 Response:** The variable actions the FCS takes when an event happens. A response can be declared for external events and for events related to accept phase. Internal responses can be declared and are referred to by event-driven responses.
- 2.109 Runtime:** The point in time at which the FCS executes forms. Compare Translation Time.
- 2.110 Run Unit:** A collection of routines and forms that are invoked at runtime by the operating environment.
- 2.111 Scalar:** A single element, as opposed to multiple elements
- 2.112 Scrolling:** Altering the display of a portion of an array so that one or more entire rows or columns of occurrences are removed from one edge of the array image, all other visible occurrences are moved toward that edge, and one or more new occurrences appear at the opposite edge of the scrolled area.
- 2.113 Selected Bar Menu:** A bar menu that will be used for a panel when requested by the ENTER BAR MENU response step.
- 2.114 Selected Transient Menu:** A transient menu that will be used as the popup menu for a panel is requested by the ENTER TRANSIENT MENU response step.
- 2.115 Send Shadow Packet:** A message sent from the program containing one of three types of information about data in the corresponding data packet: 1) an indicator specifying whether the packet fields in the associated send packet should change the last-known value of form data items changed by the send packet; 2) indicators for each packet field in the associated send packet specifying whether the packet field is invalid; 3) property change information for form data associated with the packet fields, including protection level and highlight level. A send shadow packet contains one character for each packet field in the send packet plus one additional character.
- 2.116 Session:** The connection between a program and a terminal using a form. A session is made up of the named terminal, the named form, and the state of form data, and display objects. The FCS assigns a session identification string for each session. This string consists of sixteen characters, the meaning of which is implementor defined.
- 2.117 Shadow Packet:** A message containing information about the packet fields in a packet. See Send Shadow Packet and Receive Shadow Packet.
- 2.118 Sibling Object:** A display object that is a child of the same parent as another display object.
- 2.119 Simultaneous Request:** The situation in which the FCS has received another request before completing a first request.
- 2.120 Style Property:** An IFDL clause in a display object; the clause's properties may be interpreted by the implementation to match a style guide for the implementation or the capabilities of the terminal.
- 2.121 Style Object:** An IFDL display object whose properties may be interpreted by the implementation to match a style guide for the implementation or the capabilities of the terminal.
- 2.122 Synchronous Request:** A request that shall complete before the FCS returns control to the calling program.
- 2.123 Terminal:** A device consisting of a display device and optional input devices.
- 2.124 Thread:** A line of control of execution within a process.
- 2.125 Tracked Attribute:** An attribute of form data that directs the FCS to monitor the value of the data and be prepared to report any changed status in a receive shadow packet.
- 2.126 Translation Time:** The point in time at the syntax of a form is checked. Compare to Runtime.
- 2.127 Viewport:** A rectangular area specified as a range of lines and columns within the display.

ISO/IEC 11730:1994(E)

3 FIMS philosophy of use

The Form Interface Management System (FIMS) is a software aid for handling the human interface requirements of modern applications. As its name implies, FIMS is well-suited for providing the interface to interactive applications. It has specialized features for visual display units and alphanumeric formatting, but it can be used with a wide variety of devices, including non-visual devices using non-alphanumeric I/O. In all discussions of FIMS, all types of devices are referred to as terminals.

Terminals are complex and varied devices so that handling them is a complex task. The variety of terminal hardware is expected to continue to increase. In order to protect the large investment in current and future application programs such programs should be as immune from the effects of such changes as possible. The philosophy of FIMS is that the application program should concentrate on data processing and that terminal management should be handled separately. In particular, the application program should know as few as possible of the physical characteristics of the terminal device that it is using. An objective is to increase the level of portability and maintainability of terminal handling applications to at least the level of traditional batch applications.

In order to achieve this objective, FIMS defines an Independent Form Description Language (IFDL); this language is used to define the appearance and processing of an entity called a form, which describes the interface to a terminal. FIMS also defines a generic interface, which describes the way in which an application program interacts with the IFDL during run-time.

FIMS has two conformance classes (see 4), Basic Subset/Full and Synchronous/Asynchronous. The design principles that guided the contents of the Basic Subset/Full class follows.

3.1 Design principles for the Full FIMS

The following are the principles used in the design of the Full FIMS. They are ranked in descending priority order. The Full FIMS attempts to satisfy them all.

Ease of Use

The Full FIMS caters to three types of user:

- a) the form designer;
- b) the programmer (may or may not be the same person as (a));
- c) the end user, called the operator in this International Standard.

The syntax and semantics of the Full FIMS are important for types (1) and (2). The range of facilities that the Full FIMS provides are important to type (3). These facilities permit the design of terminal interactions that are easy to use.

Terminal Independence

The Full FIMS provides facilities to allow the application program to be independent of the physical terminal being used.

In particular, this implies that the Full FIMS has capabilities outside of the program to match the kind of terminal that the form was designed for and the kind of terminal that is currently being used. The Full FIMS has facilities for accepting and displaying formatted data of a size other than that of the physical display being used.

It is possible to extend the form description facilities easily as terminal facilities evolve. Any extensions should have no impact on any application programs that do not require them.

Flexibility of Control

The Full FIMS contains facilities for exercising flexible control over as wide a range of terminals as possible, consistent with ease of use, state of the art, and maximum utility for the majority of users. The application can exercise as fine or as coarse a control over the physical terminal as it wishes and as is consistent with the other requirements.

Language Independence

The capabilities of the Full FIMS is usable by programs written in a wide variety of modern languages.

Distribution

The Full FIMS permits its functions to be distributed as far as any implementation allows, for example, among different host computers in a network or to intelligent terminals. Also, subsetting or leveling is possible in standardization and implementation.

Validation

The Full FIMS permits the programmer/form designer to specify criteria for deciding whether an input value is valid. In particular, this validation is distributable.

Multiple Sessions

A program using the Full FIMS can have more than one active session concurrently, where session is defined as the current state of a dialog with an operator.

Compatibility with other developments

The definition of the Full FIMS should be compatible where necessary and possible with other developments and standardization efforts.

3.2 Design principles for Basic Subset FIMS

The goals of Basic Subset FIMS are

- to expedite usable implementations of FIMS.
- to define a form management system that is much simpler than the Full FIMS yet is upwards compatible.
- to retain the chief characteristics of the Full FIMS to as large an extent as is possible given the other goals. In particular, the characteristics seen to be most important are
 - 1) terminal Independence for the application program.
 - 2) language Independence for the programming interface.

In general, Basic Subset FIMS modifies the requirements for the full FIMS by expecting the application to be much more aware of the precise nature of the terminal and to be in a greater degree of control of the user interaction. Basic Subset FIMS modifies the design principles listed above as follows:

Upwards Compatibility

An application (program and form) developed for Basic Subset FIMS can run unchanged with an implementation of the Full FIMS.

Ease of Use

As for Full FIMS.

Terminal Independence

The application program is less device independent with Basic Subset FIMS than it is with the Full FIMS. The application program does not contain details of the hardware characteristics of the device but it is more

sensitive to the flow control of the user interface than it would be with the Full FIMS.

Flexibility of Control

There is very little control that can be specified in the form. The application program will therefore be more involved in the details of the interaction with the operator.

Language Independence

As for Full FIMS.

Distribution

The application may be less efficient in a distributed environment than if it were using the Full FIMS.

Validation

The application program will be more involved in the validation of the data entered. In particular, only format and some content validation is done by the form. Cross-field and context validation is done by the program.

Multiple Sessions

As for Full FIMS.

Compatibility with Other Developments

As for Full FIMS.

3.3 FIMS model

The model of interaction with a terminal that FIMS uses consists of at least the following:

- 1) The *terminal*, a device connected either directly or via a communication line.
- 2) The *form control system (FCS)* which controls the activity on the terminal.
- 3) An *application program interface (API)* to the FCS that can be called by application programs. Alternately, in Full FIMS, the FCS can be started with a form as a main program.
- 4) A *form*, which describes terminal interactions; forms are described in the Independent Form Description Language (IFDL). In general, the form is used for interface purposes and not for an entire application. For example, in FIMS, the form does not have explicit syntax for access to files and does not have extensive calculational ability.

- 5) The *program*. In some cases a separate application program written in a third or fourth generation language acts as the main program, calling the FCS to invoke the activity of the form as the user interface. Alternately, in Full FIMS, the form may itself act as the main program, calling on routines to perform some activities; those routines, in turn, may then call on the FCS to invoke further activities. The program may contain both application records

(called packets) that are transferred to and from the form, and form request statements that initiate form activity.

- 6) The *operating environment*

The relationship of these items is indicated in Figure 1 and each is described in further detail in the clauses below.

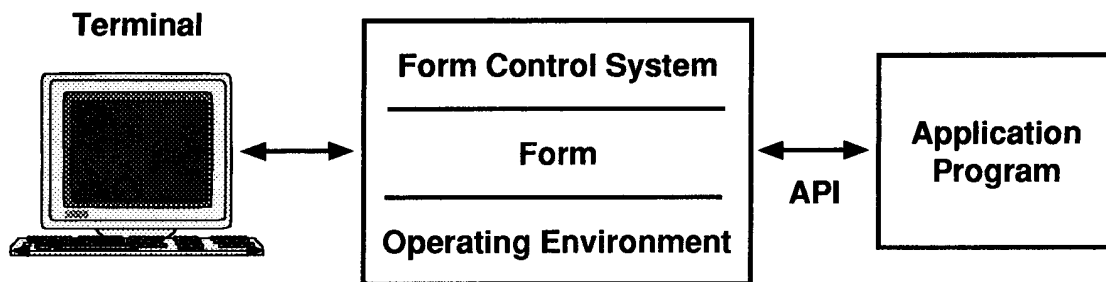


Figure 1
FIMS Model

The program's view of the terminal is through the FCS. The FCS contains the form that describes the terminal appearance and the processing of the terminal interactions. The transformation from the program's view to the physical terminal is defined by the form. The program only directly manipulates packets containing data. The relationship of this data to what is viewed on the display is defined in the form.

3.3.1 The terminal

A terminal consists of an optional display device and optional input devices.

The display device is any type of output device, for example, video display unit, hard copy terminal, printer, or voice output. Special display objects are defined in FIMS for two dimensional devices and especially for video display units. Full FIMS has special display objects in the form for dealing with "batch" devices, that is, devices that are not interactive; in this case, the form is essentially a tool for hard copy output.

In the simplest case for video terminals (Figure 2), the display occupies exactly the entire display device, no more and no less.

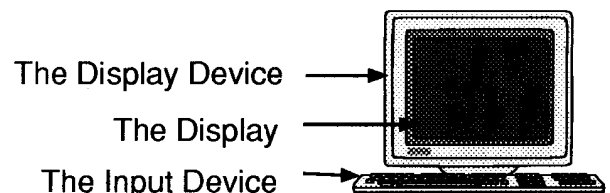


Figure 2
Device Terminology, Simple Case

In a more complex situation, the operating environment may allow the display device to be used by one or more programs and may provide its own control over positioning of the display on the display device for each program (Figure 3).

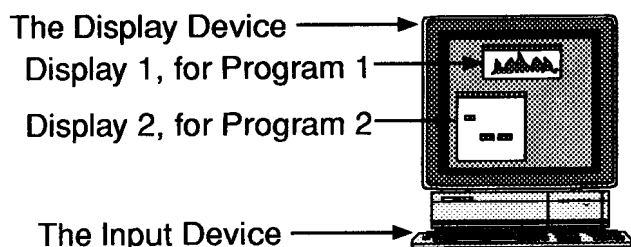


Figure 3
Multiple Displays on a Display Device

When more than one program uses a display device, the operating environment hides the interactions from the programs.

On the display for a video device, the form can choose to divide the display into one or more areas called viewports to provide a logical segmentation of the information provided; portions of the form can be mapped to different viewports (Figure 4). In Full FIMS, the default viewport is the entire display; Basic Subset FIMS has no default viewport—each viewport is declared explicitly.

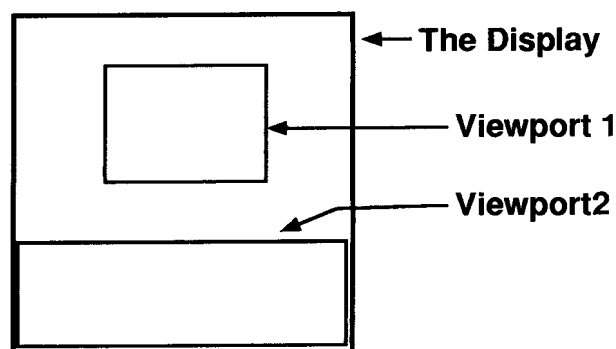


Figure 4
Viewports on a Display

The three terms, display device, display, and viewport, are required to handle the variety of cases possible. In the simplest case, the display device, the display, and the viewport are identical in size, but in other cases they can be different. (Figure 5).

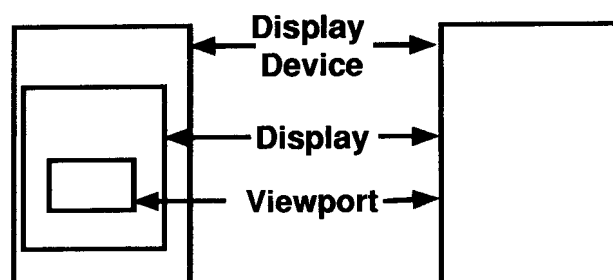


Figure 5
Simple and Complex Sizes

The FIMS model specifies that the display be no larger than the display device and that viewports be no larger than the display. The FIMS model specifies that the display be no larger than the display device and that the dynamic size of viewports be no larger than the display. There is a provision for elements displayed in viewports (called panels) to be larger than can fit on the viewport. In this case, the implementation may provide the panning of a window over the panel in any way it wishes, with the provision that the form not be aware of it.

For Basic Subset, panels fit in viewports; that is, there is no panning required.

FIMS considers three classes of input devices, depending on the kind of information that the device can deliver. These are known as the function key class, the string class, and the locator class. An input device of function key class returns only a signal identifying the device itself; an example of such an input device is a single function key. An input device of string class returns a character string, in addition to the device identification; the string input device most commonly used is a typewriter-style keyboard; other examples are badge reader and voice. An input device of locator class returns a location on the display area, in addition to the device identification; examples are mouse, light pen, joystick, and touch panel.

3.3.2 The form control system (FCS)

The form control system handles all interactions with the terminal under control of the form description. It also handles the transfer and processing of data between the program and the terminal.

In order to interact with the FCS, a program declares data areas and calls the FCS using those data areas, or in Full FIMS, it may respond to calls from the FCS using those areas. When it acts a calling program, the program declares the data areas and invokes the FCS functions in the context of a particular programming language, called the host language. The host language might or might not have special syntax to aid in FCS interaction. Whatever elements of the host language are used for the data declarations are called the application declarations. Whatever procedural elements of the host language are used for initiating FCS interaction are called the form request statements.

In Basic Subset FIMS, the same form request statements are defined as in the Full FIMS. However, an application will make more form requests using Basic Subset FIMS than it would when using the Full FIMS because Basic Subset FIMS contains a smaller set of procedural constructs than the Full FIMS, requiring the program to execute more procedural control over the interaction.

The FCS processes requests from the program, using both application declarations and Independent Form Description Language specifications. The FCS passes status information back to the program as appropriate to the request.

The FCS can be implemented in a number of different ways. For example, it can be one physical program within the operating environment. Alternatively, the functions of the FCS can be provided by already existing operating environment code or by code within the terminal.

Details of the FCS actions are found in Clause 7, FCS processing model.

3.3.3 The application program interface

The FCS is called from programs by means of the form request statements and application declarations of the language in which the program is written. There are many languages in use today. It is not possible to provide a single specification for all languages. FIMS specifies, instead, a generic interface, a definition of what kind of information the form request statements and application declarations deal with and what every FCS handles. Each implementation of the FCS shall provide the services defined in the generic interface. The exact details of an implementation interface

depend on operating environment conventions and other environmental considerations.

Using the generic interface specification as a guideline, language designers may choose to design special syntax to enable programs to invoke the FCS in a convenient manner. For each implementation of the language, the special syntax translates into invocations of the FCS's implementation interface. Alternately, language designers may choose to require a language user to use the implementation interface directly, using the language's native facilities. Clause 6, Language bindings, contains guidelines for language bindings.

There are five primary requests in the generic interface for synchronous implementations: ENABLE, DISABLE, SEND, RECEIVE, and TRANSCEIVE. For asynchronous implementations there are two additional requests: CANCEL and POLL.

The ENABLE function connects the program to the terminal and selects a form, initiating a session. The SEND function causes zero, one or more record messages, also called packets, to be transferred from the program to the form. The RECEIVE function causes one or more packets to be transferred from the form to the program. The TRANSCEIVE function causes, first, packets to be transferred from the program to the form and, secondly, packets to be transferred from the form to the program. The DISABLE function is used to indicate that the program is finished with the form and with the terminal, terminating the session.

The CANCEL function stops the processing of some requests. The POLL function asks for information on the completion status of some requests.

3.3.4 The form

The form is the central concept within FIMS for terminal interaction. It is created by statements in the Independent Form Description Language (IFDL). This language is designed to describe the functions that are required for terminal interaction.

The IFDL also provides the base description of all packets that are exchanged between the application program and the form. This International Standard does not specify any translation between a description of a packet in another format and the IFDL format. The IFDL format is defined such that translation is possible from IFDL to a wide variety of host language formats and vice versa. It is an option for implementors to provide tools to allow use of a IFDL description from a host language and the use of a host language description from IFDL.

The form contains all the terminal-dependent information within FIMS. It describes all the terminal interactions together with the data that is transferred to and from the application program and any processing that is to occur. Therefore the program is independent of terminal requirements.

Further concepts are explained in Annex A, IFDL Philosophy of Use.

3.3.5 The application program

The application source program contains two components for form management:

- 1) *Application Declarations*
The application declarations are a subset of the programming language's data description syntax. They describe the format of the packets and other information that pass between the program and the form.
- 2) *Form Request Statements*
Form request statements initiate activity on the display through the FCS. They can consist of either subroutine calls or specific syntax that generates the appropriate invocations of the FCS via the generic interface.

3.3.6 The operating environment

The operating environment consists of the operating system services and utilities that can be used by an implementation to achieve the specifications of FIMS. FIMS does not specify any specific services or utilities, so that an implementation can use or invent whatever it needs to achieve its effects.

3.4 Relationships among display, form, and program

A session includes the connection between a form and device. In Full FIMS, a session can be started by the operating system or by an application program. In Basic Subset FIMS, a session is started only by an application program. Figure 6 shows the two interfaces in a session. A session started by an application program commences when the form request statement ENABLE connects the program to a terminal through a form. Thereafter, a dialogue ensues using the form request statements SEND, RECEIVE, TRANSCIVE, CANCEL, and POLL. A session terminates when the form request statement DISABLE is used to break the connection between the program and the terminal.

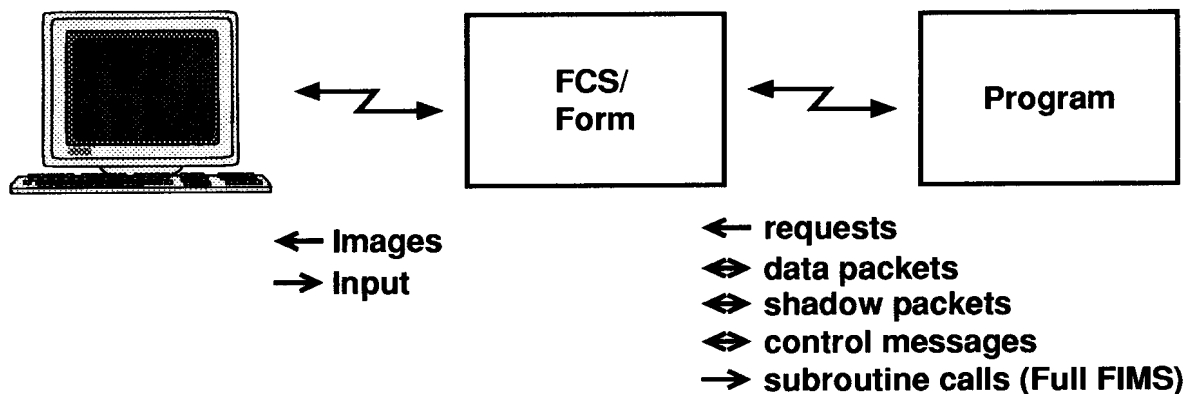


Figure 6
The Two Interfaces in FIMS

Execution control is passed from the program to the FCS which returns it back again after the request completes. In Full FIMS, execution Execution> control is passed from the FCS to subroutines (specified in the form) in the program which then return to the FCS.

Data is considered to flow between the program and the form in messages called packets and control messages. Image messages pass between the display and the form.

Packets are described by both the IFDL and the application declarations. There are two types of packets, data packets and shadow packets. Data packets correspond to application program data records; they are made up of characters or binary data. Shadow packets correspond to application program records that contain information *about* the fields within the data packets.

A control message is composed of strings of characters. Control messages sent to the form are for the purpose of generating special control actions in the form agreed upon between the form designer and the programmer. Control messages sent to the program report on the status of form request statements and on special conditions the form designer wishes to have reported to the program.

The translation of the IFDL and the program may each occur at different times. In addition, the binding of the IFDL and the program may also occur at different times.

3.5 Use of FIMS

The Form Interface Management System is designed for use in a wide variety of environments. By standardizing the program interface to user interaction, the computational code part of an application is relieved from having detailed or even general knowledge of the device being used.

More importantly, by isolating the user interface into the entity known as the form, implementations have a great deal of freedom in helping interface designers and in being creative themselves. The IFDL representation of the form is one method of describing the user interface. As a text based language, it is precise, is standardized, and can be ported from one system to the next.

Most implementors may, in addition, provide a richer development environment than just a translator for IFDL. It is recommended that, at the very least, most implementations should provide interactive screen editors (also called screen painters) that the form designer can use to describe the appearance of the form on the screen. The equivalent IFDL would then provide an implementation-independent representation of a screen description that was generated from an implementor-defined, internal screen definition.

A close analogy can be made between the IFDL part of FIMS and word processing/document preparation. Each word processing package keeps its own internal format and provides its own user interface to an operator who is creating a document. At first, different WP packages could not read files created by other packages. Then interchange formats were developed, some of which are undergoing standardization at this time. The interchange formats were at first simple (the first was simply ASCII) but became more complex as time went on. Users did not want to lose information merely because they were going through an interchange. As a result, some of the interchange formats became more complex; for example, some contain provision for graphics directly, some for encapsulated graphics, and so forth. The individual WP packages

then became richer as they wished more fully to support the more complete interchange formats. At some point, the interchange format becomes the standard, while the individual packages continue to provide unique development services.

IFDL fills the roles of the interchange format in this analogy. Its intent is to provide a rich basis for portable user interface definitions for predefined dialogs (forms). User interface editors (screen editors, screen painters, dialog generators, and so forth) are individual implementations' ways of making the creation process easier for form designers. Such editors often generate data aggregates and files that are directly usable by the FCS at run time. Alternately, they may produce the IFDL equivalent for direct usage. Of course, the IFDL equivalent is required for portability.

In addition, other utilities may produce FIMS compliant interfaces because of the flexibility it gives them. Many 4GLs (4th Generation Languages—typically application generators) have their own forms systems because there is yet no International Standard. Just as many 4GLs are capable of taking advantage of data base access standards to get to implementors' unique data bases, they can also take advantage of a forms access International Standard to get to implementors' unique terminal handlers. One of the major features of a 4GL is integration with a data dictionary, usually vendor specific. The 4GL provides access to such a dictionary and could create IFDL form descriptions that are consistent with that dictionary.

For the 3GL (3rd Generation Language—procedural languages such as C, Fortran, and COBOL) case where there is a non-integrated data dictionary capability, IFDL provides the capability of retrieving definitions of data and packets by the COPY statement. This allows consistency of definition of packets between application programs and forms as well as enforcing standards across the organization for field definitions. The extent of this ability is implementor defined at this time.

Figure 7 shows a possible implementation strategy. Any ways of creating IFDL forms and linking them into the run-time system are equivalent as far as this International Standard is concerned.

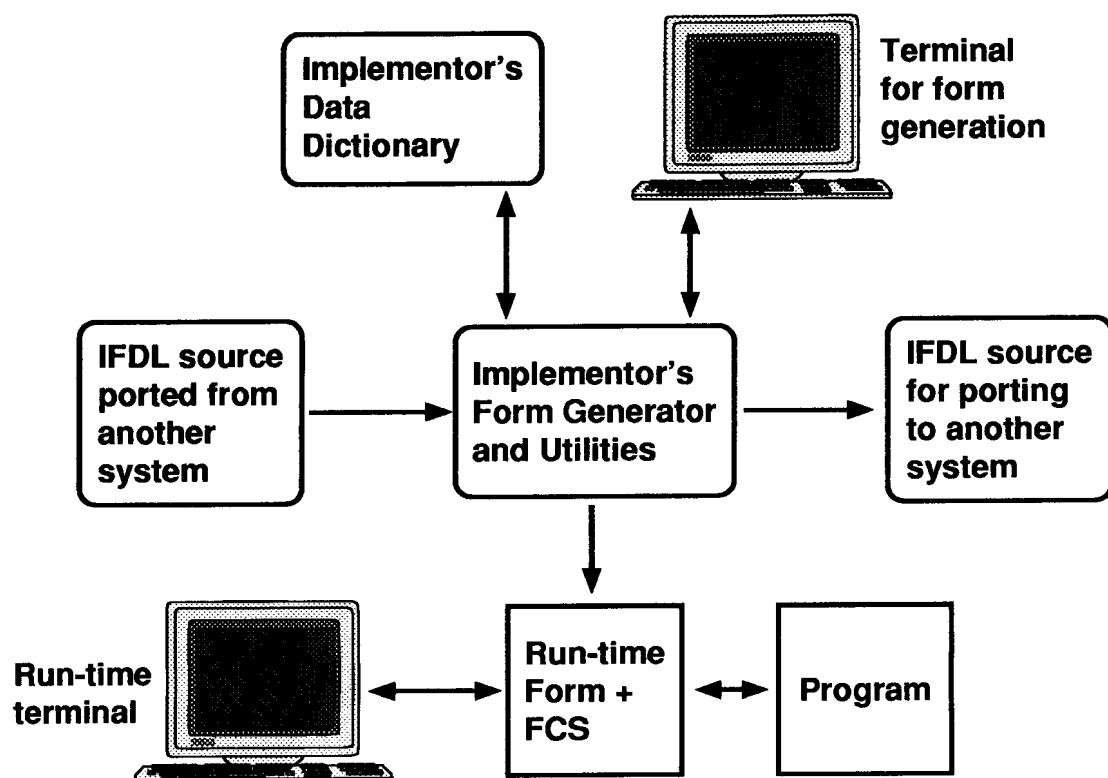


Figure 7
A Possible Implementation of FIMS

Section 2: Conformance

The FIMS is a complex system, consisting of two major categories of two levels each of capabilities. Because of the wide variety of terminal capabilities, there are a large number of implementation options that state what an implementation provides. The rules that govern whether an implementation conforms to the FIMS International Standard are stated in Clause 4, Conformance.

4 Conformance

This clause describes conformance of an implementation to the ISO/IEC 11730.

4.1 Categories

There are two categories describing the level of conformance of a FIMS implementation:

- 1) Basic Subset/Full
- 2) Synchrony/Asynchrony

There are ten categories of implementor defined features.

- 1) Device Types
- 2) Panning
- 3) Literal Types
- 4) Presentation Attributes
- 5) Culture
- 6) Data input
- 7) Locator input
- 8) Implementor names
- 9) Style elements
- 10) External panels

A complete specification of the levels of conformance, of the implementor defined features, and of the language bindings supported constitute a profile for the implementation.

4.2 Conformance

An implementation can be conformant at any the four levels resulting from combinations of the two conformance categories, each of which has two levels.

4.2.1 Basic subset/full

An implementation conforms to Basic Subset FIMS if it implements those IFDL syntax features and generic interface features identified as belonging to Basic Subset FIMS.

It conforms to Full FIMS if it implements those syntax features and generic interface features identified as belonging to Full FIMS.

Basic Subset FIMS is a proper subset of Full FIMS, so that everything that applies to Basic Subset FIMS also is true of the Full FIMS.

This International Standard includes information about both the Full FIMS and Basic Subset FIMS.

Since the International Standard contains information about both levels, there is a way to determine what level a given statement refers to. The following principles are followed in specifying this.

- 1) In many cases, a whole clause of the International Standard will apply only to Full FIMS and not to Basic Subset FIMS. In this case, a statement to that effect is printed at the beginning of the clause: "This clause applies only to the Full FIMS."
- 2) In a clause that applies to both Full and Basic Subset FIMS, an occasional single paragraph will be labeled: "This paragraph applies only to the Full FIMS."
- 3) There are times when the subset behavior is most easily called out explicitly, since it is a major simplification of the full behavior and might not be immediately obvious; such cases always explicitly mention the subset.

4.2.2 Synchronous/asynchronous

An implementation conforms to synchronous FIMS when it implements those generic interface features that support synchronous requests. If, in addition, it implements those generic interface features that support asynchronous requests, it conforms to asynchronous FIMS.

Both Basic Subset FIMS and Full FIMS implementations that do not conform to asynchronous FIMS shall implement the asynchronous requests POLL and CANCEL, but such requests return only an error control text. The other five requests provide for asynchronous operation; but synchronous conformance need only service the requests synchronously.

4.3 Profile

None of the implementor defined features are required at runtime for conformance, but an implementation shall be prepared to deal with the syntax specifying the feature, even if it only ignores the syntax without error. An implementation shall state what it does in each category.

4.3.1 Device types

An implementation of FIMS shall be deemed conforming if it supports at least one of the device profiles listed in Clause 10.

An implementation shall list the devices types it supports. It shall recognize as a conforming form description any form that provides correct syntax in the device-declaration of a layout. If the implementation does not support that device type or some of its attributes, it shall perform one of two actions:

- Provide an implementor-defined fallback mechanism for the attributes it does not support.
- Ignore the layout.

4.3.2 Panning

Conformance does not require that an implementation provide panning. Specifically, an implementation can be conforming if the size of panels is restricted to the size of the associated viewport. If an implementation allows panning (that is, it allows panels larger than viewports), it shall provide the panning by use of the scroll bar syntax element or some implementor defined manner.

4.3.3 Literal types

Not all terminals are capable of representing all the literal types available in IFDL syntax.

Implementations for terminals that are character based need support only literals of type text, rectangle and those polyline literals that consist of only vertical and horizontal lines.

Implementations for terminals that are pixel based are expected to support polyline and arc literals. It is an implementation choice to support raster and import literals. If import literals are supported, the implementation shall list the formats accepted and the compression type recognized.

In all cases, an implementation shall recognize without error all syntax in the International Standard. If an implementation does not support a literal type, it shall perform one of two actions:

- Provide an implementor-defined fallback mechanism for the literals it does not support.
- Ignore the syntax.

4.3.4 Presentation attributes

Presentation attributes are represented by the set of attributes in the display-xxx syntax elements (display-item, display-viewport, and so forth); in turn, these syntaxes reference other clauses. A conforming implementation shall recognize all such clauses without error, even if it does not support such presentation attributes.

If an implementation does not support a presentation attribute, it shall perform one of two actions:

- Provide an implementor-defined fallback mechanism for the attributes it does not support.
- Ignore the attribute.

4.3.5 Culture

Culture includes the language path display attribute (also called writing direction), the clauses of the culture-declaration, and character sets.

- An implementation shall state which language paths it supports and the implementation defaults for the culture clause.
- If an implementation does not support a language path, it shall provide an implementor-defined fallback for the writing directions it does not support.
- An implementation shall state which character sets it supports.

4.3.6 Data input

There are several places in field input processing where implementor options are specified:

- Intra-field editing functions in function-declaration
- Multiple key sequences in function-declaration
- INPUT JUSTIFICATION

In all cases, an implementation shall recognize without error all syntax specified. If an implementation does not support a data input option, it shall perform one of two actions:

- Provide an implementor-defined fallback mechanism for the attributes it does not support.
- Ignore the option.

4.3.7 Locator input

An implementor may choose to support locator input for a terminal. If it does not, it shall still recognize without error all syntax referring to the locator. If an implementation does not support locators for a terminal, it shall perform one of two actions:

- Provide an implementor-defined fallback mechanism for the locator features it does not support.
- Ignore the locator feature.

4.3.8 Implementor names

Implementor names appear in several places in the syntax. (They are identified in the index.) An implementation shall recognize without error all legally-formed implementor-names. If an implementation does not support the particular implementor-name encountered, it shall perform one of two actions:

- Provide an implementor-defined fallback mechanism for the syntax element in which it appears.
- Ignore the implementor-name.

4.3.9 Style elements

Several objects of both Full and Basic Subset IFDL syntax are identified as "style objects". All style objects shall be present in an implementation, but the implementation need not provide appearance and behavior corresponding to every clause of the style object. In addition, some clauses in objects not identified as style objects may be "style properties" on their own. If an implementation does not support a style property or style object, it shall perform one of two actions:

- Provide an implementor-defined fallback appearance and behavior for the clause or object.
- Ignore the clause without error.

In addition, those elements of appearance and behavior not specified in the IFDL declaration for style objects are free to be implementor defined.

The intent of this conformance rule is to give implementations as much flexibility as possible to create their own look and feel for the style elements while allowing for the widest possible applicability of a single syntax to a variety of interface styles.

The style objects and properties are identified as such in the semantic rules for IFDL statements.

4.3.10 External panels

An implementation need not support external panels. If it does, it shall state what devices it supports them for and what the calling sequence parameters are for procedural escapes associated with the external panel.

If the implementation does not support external panels, it shall recognize valid syntax associated with an external panel without error.

Section 3: Application interface

The generic interface (Clause 5, Generic interface) is an abstract definition of application programs can invoke the forms defined in IFDL. The generic interface is not a specification for a particular programming language. It does, however, specify what information shall be included in such a specification. Clause 6, Language bindings, describes the state of some language bindings.

5 Generic interface

5.1 Introduction

The FIMS contains a generic programming interface to the FCS, describing in general terms what procedural elements and data structures are needed for a program to invoke FCS. While this generic interface is specified most carefully, it is not what most application programmers will see. Rather, the generic interface defines what shall be provided in any programming language that interfaces to the FCS. A particular programming language will define a specific interface for users of that language. Such a specific interface is called the *binding* or the *application programming interface* of the FCS generic interface to that language.

The generic interface does not prescribe a way of passing additional, optional, implementation dependent parameters to the FCS. Use of such parameters makes a program less portable, but is not forbidden by this International Standard. A language binding shall provide a method for passing such parameters.

The generic program interface for Basic Subset FIMS is upwards compatible with the full FIMS. That is, if a request is successful in Basic Subset FIMS, it will also be successful with the same effect in full FIMS.

5.2 Request descriptions

There are seven requests that an application program can make of the FCS. Two requests are specifically for asynchronous support, while the other five can be used either synchronously or asynchronously. Each of these requests usually corresponds to a form request statement in a host language (either a subroutine call or explicit language syntax for FIMS).

Each request has parameters and produces an action in the FCS. Each request is described in terms of its parameters, what action it produces, and the errors that might arise from the request.

Since several requests have the same parameters, the parameters are described in Table 1. For each request, three pieces of information are supplied for each parameter:

- Parameter name

- Direction of information transfer, one of AP $\Rightarrow$ FCS (from the application to the FCS), FCS $\Rightarrow$ AP (from the FCS to the application), or FCS $\leftrightarrow$ AP (from the application to the FCS and also from the FCS to the application)
- Requirements concerning the parameter: when it shall be present and the FCS action if the parameter is absent, that is, it has a null value

If a parameter is labeled "shall appear" or "shall meet some condition", the FCS shall enforce that condition and reject the request if the parameter does not appear or does not meet the condition.

Some parameters are not used in Basic Subset FIMS and are so labeled.

Potential errors for the requests are described in Table 2.

In this generic specification, errors are reported in the receive-control-message, a parameter to the requests. If the receive-control-text-count parameter is set to zero by the FCS on return from the request, the request has been completed successfully. If the receive-control-text-count parameter is not zero, the FCS is returning significant information in one of the control texts in the receive-control-message parameter; the application examines those control texts to determine whether the request completed successfully. If a control text begins with a blank or with "K", the control text is merely reporting useful information. If a control text begins with "E", the control text contains error information. Error information means that the state of the session might no longer be valid.

In language bindings, errors may be additionally be reported in some other fashion, usually as the result of the call (that is, as a function value).

5.2 Request descriptions

5.3 Cancel request

Cancel Request Parameters

Parameter name	Direction	Requirements
Request-id	AP⇒FCS	Either request-id or session-id shall be specified. The other shall be null.
Session-id	AP⇒FCS	Either request-id or session-id shall be specified. The other shall be null.
Receive-control-message	FCS⇒AP	Null means no control message is returned to the program.
Receive-control-text-count	FCS⇒AP	Null means no control message is returned to the program.

FCS action

For synchronous FIMS implementations, the cancel request does nothing, and returns an ES020 error code.

The cancel request aborts one or more Enable, Disable, Send, Receive, or Transceive requests. It cancels the particular request specified by request-id or all requests associated with session-id. In addition, it cancels all descendants of the requests specified. A descendant request is one that specified a parent-request-id matching the request-id of the canceled request.

If a request-id is not valid (including the case where a the request completed), an error is reported (ES019).

The cancel request itself returns immediately to the calling program; its receive-control-message specifies the success or failure of the cancel request itself.

All requests canceled by the the cancel request complete at a later time with an error status (ES017); they execute no further responses and return no packets. The session data and the display device are left in an indeterminate state.

A cancel request cannot be canceled. If another cancel request for the same session or request-id is given to the FCS, the second cancel request returns immediately will an _S002 control text to indicate that the second cancel request is not needed.

Errors

ES005 ES011 ES019 ES020 ES031 _S002

5.4 Disable request

Disable Request Parameters

Parameter name	Direction	Requirements
Session-id	AP⇒FCS	Shall not be null.
Receive-control-message	FCS⇒AP	Null means no control message is returned to the program.
Receive-control-text-count	FCS⇒AP	Null means no control message is returned to the program.
Send-control-message	AP⇒FCS	Null means no control message is sent to the form.
Send-control-text-count	AP⇒FCS	Null means no control message is sent to the form.
Timeout	AP⇒FCS	Null means no timeout is applied.
Request-id	FCS⇒AP	Null means the request is synchronous and no request-id will be returned to the program.
Parent-request-id	AP⇒FCS	Null means this request is not associated with a previous request. Not used in Basic Subset FIMS.

FCS action

The disable request terminates the session. The FCS first performs disable external request processing as defined in the form. The form attached to the terminal is unloaded and the session information is lost. No further requests can be made of this session.

Errors

ES001 ES005 ES010 ES011 ES012 ES013 ES014 ES017
 ES020 ES021 ES023 ES024 ES025 ES026 ES027 ES028
 ES031 _S001

5.5 Enable request

Enable Request Parameters

Parameter name	Direction	Requirements
Culture-string	AP⇒FCS	Null or blank means the culture string is supplied by the environment.
Form-name	AP⇒FCS	One of Form-name and Form-library-name shall not be null.
Form-library-length	AP⇒FCS	Shall be specified when form-library-name is specified.
Form-library-name	AP⇒FCS	One of Form-name and Form-library-name shall not be null.
Language-string	AP⇒FCS	Null or blank means the language string is supplied by the environment.
Layout-selection-string	AP⇒FCS	Null or blank means the layout selection string is supplied by the environment.
Terminal-identifier	AP⇒FCS	Null or blank means the terminal is supplied by the environment.
Terminal-length	AP⇒FCS	Shall be specified when terminal-identifier is specified.
Session-id	FCS⇒AP	Shall not be null.
Receive-control-message	FCS⇒AP	Null means no control message is returned to the program.
Receive-control-text-count	FCS⇒AP	Null means no control message is returned to the program.
Send-control-message	AP⇒FCS	Null means no control message is sent to the form.
Send-control-text-count	AP⇒FCS	Null means no control message is sent to the form.
Timeout	AP⇒FCS	Null means no timeout is applied.
Request-id	FCS⇒AP	Null means the request is synchronous and no request-id will be returned to the program.
Parent-request-id	AP⇒FCS	Null means this request is not associated with a previous request. Not used in Basic Subset FIMS.

FCS action

The FCS locates the terminal in an implementor defined fashion. Using Form-name and Form-library-name in an implementor defined fashion, the FCS locates the form. Using culture-string, language-string, layout-selection-string and the device declaration in the form, the FCS finds an appropriate layout for the terminal. If the FCS is successful, it creates a session for the specified form and terminal. The FCS creates all form data for the form and carries out enable external request processing as specified by the form. It returns a unique session identification string that is used by subsequent requests to identify this form on the specified terminal.

Errors

ES001 ES002 ES003 ES004 ES005 ES012 ES013 ES014
ES017 ES020 ES021 ES023 ES024 ES025 ES026 ES027
ES028 ES031 _S001

5.6 Poll request

The action of the poll request is not supported in synchronous-only FIMS implementations; none of the parameters are validated.

Poll Request Parameters

Parameter name	Direction	Requirements
Request-id-array	AP⇒FCS	Shall not be null.
Request-id-count	AP⇒FCS	Null means only one entry in request-id-array.
Poll-timeout	AP⇒FCS	Null means a poll-timeout of zero.
Receive-control-message	FCS⇒AP	Null means no control message is returned to the program.
Receive-control-text-count	FCS⇒AP	Null means no control message is returned to the program.

FCS action

For synchronous FIMS implementations, the poll request does nothing, and returns an ES020 error code.

The POLL request passes the request-id-array to the FCS to await completion of at least one of the requests that were started in an asynchronous manner. The FCS supplied a request-id so that the application program could determine that the asynchronous request started properly, and returned to the application program before the request was completed. The application program uses request-id to determine when the request completes.

A request is complete when it succeeds, fails, times out, or is canceled. Whenever a request completes, the FCS sets its request id to the "completed" value, a string with "1" as the first character. In the poll request, the FCS checks for a "completed" value for any request-id in request-id-array.

If no request-id in request-id-array has completed (that is, its first character is not "1") and if no request-id in request-id-array is currently valid (a valid request-id is one that will complete in the future), the FCS returns an error (ES018).

If any request-id in request-id-array is complete at the time of the poll request, the FCS returns to the calling program immediately, with success.

If no request-id in request-id-array is complete, but some request-id is valid, the FCS actions are determined by the value of the poll-timeout parameter. A negative poll-timeout value specifies that the FCS will wait indefinitely until at least one request-id in request-id-array completes. A zero poll-timeout value specifies that the FCS will return to the application immediately with the timeout error text (ES001). A positive poll-timeout value specifies that the FCS will return to the application program if no request-id in request-id-array completes by the end of the poll-timeout

period. In the case that no request completes, whatever the poll-timeout waited, a timeout error text (ES001) is returned.

The receive-control-message represents the POLL request status, not the status of the request that completed.

Errors

ES001 ES018 ES020 ES031

5.6 Poll request

5.7 Receive request

Receive Request Parameters

Parameter name	Direction	Requirements
Session-id	AP⇒FCS	Shall not be null.
Receive-control-message	FCS⇒AP	Null means no control message is returned to the program.
Receive-control-text-count	FCS⇒AP	Null means no control message is returned to the program.
Send-control-message	AP⇒FCS	Null means no control message is sent to the form.
Send-control-text-count	AP⇒FCS	Null means no control message is sent to the form.
Timeout	AP⇒FCS	Null means no timeout is applied.
Request-id	FCS⇒AP	Null means the request is synchronous and no request-id will be returned to the program.
Parent-request-id	AP⇒FCS	Null means this request is not associated with a previous request. Not used in Basic Subset FIMS.
Receive-packet-count	AP⇒FCS	Null implies one packet or shadow packet is being requested. Not used in Basic Subset FIMS, meaning that one packet is always requested.
Receive-packet-identifier	AP⇒FCS	Shall not be null.
Receive-packet	FCS⇒AP	Null means no packet will be returned to the application program.
Receive-packet-length	AP⇒FCS	Shall not be null when receive-packet is specified. Ignored when receive-packet is null.
Receive-shadow-packet	FCS⇒AP	Null means no shadow packet will be returned.
Receive-shadow-packet-length	AP⇒FCS	Shall not be null when receive-shadow-packet is specified. Ignored when receive-shadow-packet is null.

FCS action

The receive request asks the FCS to return zero or more packets and shadow packets from the form to the application program. The FCS performs receive external request processing as specified by the form. The FCS returns modified field indicators or position information in the receive-shadow-packet. If receive-packet-count is more than one, the following four parameters are repeated, once for each receive packet: receive-packet, receive-packet-length, receive-shadow-packet, and receive-shadow-packet-length. (Modified field indicators and more than one packet are not available in Basic Subset FIMS.)

Errors

ES001 ES005 ES007 ES009 ES011 ES012 ES013 ES014
 ES015 ES017 ES020 ES021 ES023 ES024 ES025 ES026
 ES027 ES028 ES029 ES031 _S001

5.8 Send request

Send Request Parameters

Parameter name	Direction	Requirements
Session-id	AP⇒FCS	Shall not be null.
Receive-control-message	FCS⇒AP	Null means no control message is returned to the program.
Receive-control-text-count	FCS⇒AP	Null means no control message is returned to the program.
Send-control-message	AP⇒FCS	Null means no control message is sent to the form.
Send-control-text-count	AP⇒FCS	Null means no control message is sent to the form.
Timeout	AP⇒FCS	Null means no timeout is applied.
Request-id	FCS⇒AP	Null means the request is synchronous and no request-id will be returned to the program.
Parent-request-id	AP⇒FCS	Null means this request is not associated with a previous request. Not used in Basic Subset FIMS.
Send-packet-count	AP⇒FCS	Null implies one packet or shadow packet is being sent. Not used in Basic Subset FIMS, meaning that one packet is always sent.
Send-packet-identifier	AP⇒FCS	Null means no packet or shadow packet is being sent.
Send-packet	AP⇒FCS	Ignored when send-packet-identifier is null. Null means no packet is being sent.
Send-packet-length	AP⇒FCS	Shall not be null if send-packet is specified. Ignored if send-packet is null.
Send-shadow-packet	AP⇒FCS	Null means no shadow packet specified. Ignored if send-packet-identifier is null.
Send-shadow-packet-length	AP⇒FCS	Shall not be null when send-shadow-packet is specified. Ignored when send-shadow-packet is null.

FCS action

The send request may pass any, all, or none of the following three types of information to the form:

- Send-control-message
- Send-packet
- Send-shadow-packet

The send request passes zero, one, or more packets and shadow packets from the application program to the form. A send-shadow-packet, if one is present, contains indicators about invalid fields or about modifying field properties, and may specify that information in the send-packet is not to change the last-known values for the packet fields. The send-shadow-packet may be sent without sending a data packet, in which case only the invalid field indicators or field properties are meaningful. If neither a send-packet nor a send-shadow-packet is sent, only the send-control-message is processed by the form. It is legal for a send request to send none of the above, in which case the FCS does nothing except determine if the session is still connected. FCS performs additional send external request processing as specified by the form.

If send-packet-count is more than one, the following four parameters are repeated, once for each send packet: send-packet, send-packet-length, send-shadow-packet, and send-shadow-packet-length. A send-packet-count greater than one is not available in Basic Subset FIMS.

Errors

ES001 ES005 ES006 ES008 ES011 ES012 ES013 ES014
ES016 ES017 ES020 ES021 ES023 ES024 ES025 ES026
ES027 ES028 ES030 ES031 _S001

5.8 Send request

5.9 Transceive request

Transceive Request Parameters

Parameter name	Direction	Requirements
Session-id	AP⇒FCS	Shall not be null.
Receive-control-message	FCS⇒AP	Null means no control message is returned to the program.
Receive-control-text-count	FCS⇒AP	Null means no control message is returned to the program.
Send-control-message	AP⇒FCS	Null means no control message is sent to the form.
Send-control-text-count	AP⇒FCS	Null means no control message is sent to the form.
Timeout	AP⇒FCS	Null means no timeout is applied.
Request-id	FCS⇒AP	Null means the request is synchronous and no request-id will be returned to the program.
Parent-request-id	AP⇒FCS	Null means this request is not associated with a previous request. Not used in Basic Subset FIMS.
Send-packet-count	AP⇒FCS	Null implies one packet is being sent. Not used in Basic Subset FIMS, meaning that one packet is always sent.
Send-packet-identifier	AP⇒FCS	Shall not be null.
Send-packet	AP⇒FCS	Null means no send-packet is specified.
Send-packet-length	AP⇒FCS	Shall not be null if send-packet is specified. Ignored if send-packet is null.
Send-shadow-packet	AP⇒FCS	Null means no send-shadow-packet is specified.
Send-shadow-packet-length	AP⇒FCS	Shall not be null when send-shadow-packet is specified. Ignored when send-shadow-packet is null.
Receive-packet-count	AP⇒FCS	Null implies one packet is being requested. Not used in Basic Subset FIMS, meaning that one packet is always requested.
Receive-packet-identifier	AP⇒FCS	Shall not be null.
Receive-packet	FCS⇒AP	Null means no receive-packet will be returned.
Receive-packet-length	AP⇒FCS	Shall not be null if receive-packet is specified. Ignored if receive-packet is null.
Receive-shadow-packet	FCS⇒AP	Null means no receive-shadow-packet will be returned.
Receive-shadow-packet-length	AP⇒FCS	Shall not be null when receive-shadow-packet is specified. Ignored when receive-shadow-packet is null.

FCS action

The transceive request first passes zero or more send-packets and send-shadow-packets from the application program to the form. Send-shadow-packet is interpreted as in the send request. The transceive request then asks the FCS to return zero or more receive-packets and receive-shadow-packets from the form to the application program. Receive-shadow-packet is interpreted as in the receive request. The FCS performs transceive external request processing as specified by the form.

If send-packet-count is more than one, the following four parameters are repeated, once for each send packet: send-packet, send-packet-length, send-shadow-packet, and send-shadow-packet-length. If receive-packet-count is more than one, the following four parameters are repeated, once for each receive packet: receive-packet, receive-packet-length,

receive-shadow-packet, and receive-shadow-packet-length. A receive-packet-count or send-packet-count greater than one is not available in Basic Subset FIMS.

Errors

ES001 ES005 ES006 ES007 ES008 ES009 ES011 ES012
ES013 ES014 ES015 ES016 ES017 ES020 ES021 ES023
ES024 ES025 ES026 ES027 ES028 ES031 _S001

5.10 Request parameters

While the FCS handles a different set of parameters in each of its requests, parameters with the same name have the same meaning for the different requests. All parameters are described in Table 1 with three pieces of information given for each parameter:

- Parameter Name: a mnemonic term used in the request descriptions
- Data Type:
 - US - unsigned short integer, at least 16 bits in length
 - L - Signed long integer, at least 32 bits in length
 - UL - Unsigned Long integer, at least 32 bits in length

- T - Text string of different lengths
- TI - Text string of length 1-30, the length of a legal identifier in IFDL
- Tn - Text string of length n
- ATn - Array of text strings, each of length n
- P - Packet
- Description

All characters in text strings are in the implementation default character set (Clause 8.2.4, Execution character sets).

For Basic Subset FIMS, some of the parameters are not used. They shall still be specified with placeholders or dummies so that programs using the subset will be compatible with full FIMS.

Table 1
Request Parameters

Parameter Name	Data Type	Description
Culture-string	TI	The culture-string is used by the FCS to select a layout in the form for the enable request. If it is blank or null, the FCS queries the environment for a culture-string.
Form-library-length	US	The form-library-length gives the length of the string containing form-library-name. The length is positive and shall be no larger than $2^{16} - 1$
Form-library-name	T	The form-library-name identifies an implementor defined collection of forms to be used in a session.
Form-name	TI	The form-name identifies the form to be used in a session. It corresponds to the form name defined in the IFDL.
Language-string	TI	The language-string is used by the FCS to select a layout in the form for the enable request. If it is blank or null, the FCS queries the environment for a language-string.
Layout-selection-string	TI	The layout-selection-string is used by the FCS to select a layout in the form for the enable request. If it is blank or null, the FCS queries the environment for a layout-selection-string.
Parent-request-id	T24	This parameter is not used in Basic Subset FIMS. The parent-request-id parameter is a request-id passed to the FCS by a new request wishing to supersede a session's active request. If that active request's id matches this parent-request-id, i.e. is the new request's parent, the new request can become the active request for the session and can gain access to the input device. The only valid use of parent-request-id is from a procedural escape unit recursively calling the FCS.
Poll-timeout	L	Poll-timeout specifies the number of seconds that the FCS is to use for determining whether and how long to wait for some event to complete during a poll request. The length shall be no larger than $2^{32} - 1$. The value may be negative (indicating indefinite wait), positive (indicating the wait time), or zero (indicating no wait).

Table 1 (Cont.)
Request Parameters

Parameter Name	Data Type	Description
Receive-control-message	T25	Receive-control-message is a single string made up of control text strings which the FCS returns to the application program to give information on the result of a request. Each control text string is five characters and there can be one to five such strings, resulting in a control message of length 5, 10, 15, 20 or 25 characters. The first character of a receive control text string is either "E", "K", or space. "E" specifies an error message, indicating that the control text contains a special condition resulting from the request. Space specifies no special condition. "K" also specifies no special condition, but is a special indicator to the application program that the control text represents a function code (which might also be called a key code, hence the "K"). The second character of a receive control text string is either "S", "I", or "F". "S" specifies that the control text arises from a condition specified in this International Standard, "I" specifies that the control text arises from a condition specified by the implementation, and "F" specifies that the control text was specified by a RETURN response step in the form. When multiple codes are reported, they are sorted by source with order (S, F, I), then by severity, with order (E, K, blank), then by collating sequence order. The control texts specified by this International Standard are listed in Table 2.
Receive-control-text-count	US	Receive-control-text-count is the number of control text strings returned by the FCS in Receive-control-message.
Receive-packet	P	Receive-packet is the packet that the application program receives from the form.
Receive-packet-count	US	This parameter is not used in Basic Subset FIMS. Receive-packet-count is the number of packets and shadow-packets being requested from the FCS by the application program. If it is not specified, the FCS assumes a count of one (1). The FCS expects this number to match the number of packets implied by receive-packet-identifier: if that is a packet-name, the FCS expects 1; if that is a packet-list-name, the FCS expects a number equal to the number of packets named in the packet list.
Receive-packet-length	UL	Receive-packet-length is the length of the data (in characters) of receive-packet. It serves as a validity check; both the application program and the form shall agree on the length of the data being received from the form. The length is positive and shall be no larger than $2^{32} - 1$.
Receive-packet-identifier	TI	Receive-packet-identifier is the name of a packet or packet-list that the application program wishes to receive from the form. It shall match the name of a packet or packet-list known in the form for the current session. If receive-packet-count is null or one, receive-packet-identifier is interpreted as a packet-name. If receive-packet-count is greater than one, receive-packet-identifier is interpreted as a packet-list-name; this packet-list-name shall match the name of a packet-list in the form for the current session.
Receive-shadow-packet	T	Receive-shadow-packet transfers information in both directions, to the form and to the program. The length of receive-shadow-packet is one plus the number of fields in receive-packet as declared in the IFDL packet-declaration. On receipt of a receive-shadow-packet, the FCS interprets its first character as follows: 1) If it is "P", position information is returned as specified below. 2) If it is "N", the modified field information is returned as specified below, but the last-known values in form data is not reset at data collection. 3) If it is anything else, the modified information is returned as specified below and the last-known values in form data are reset.