ASG
Software Solutions

ASG-MethodManager®
Dictionary/Repository Information Model

Version 2.5

Publication Number: MMR0200-25-MDRIM
Publication Date: December 2000

The information contained herein is the confidential and proprietary information of Allen Systems Group, Inc. Unauthorized use of this information and disclosure to third parties is expressly prohibited. This technical publication may not be reproduced in whole or in part, by any means, without the express written consent of Allen Systems Group, Inc.

© 1998-2002 Allen Systems Group, Inc. All rights reserved.
All names and products contained herein are the trademarks or registered trademarks of their respective holders.

ASG Worldwide Headquarters Naples Florida USA | asg.com | info@asg.com
1333 Third Avenue South, Naples, Florida 34102 USA Tel: 239.435.2200 Fax: 239.263.3692 Toll Free: 800.932.5536 (USA only)

ASG Documentation/Product Enhancement Fax Form

Please FAX comments regarding ASG products and/or documentation to (239) 263-3692.

Company Name	Telephone Number	Site ID	Contact name

Product Name/Publication	Version #	Publication Date
Product:		
Publication:		
Tape VOLSER:		

Enhancement Request:

© 2002 Allen Systems Group, Inc.
All names and products are trademarks or registered trademarks of their respective holders.

ASG Support Numbers

ASG provides support throughout the world to resolve questions or problems regarding installation, operation, or use of our products. We provide all levels of support during normal business hours and emergency support during non-business hours. To expedite response time, please follow these procedures.

Please have this information ready:

- Product name, version number, and release number
- List of any fixes currently applied
- Any alphanumeric error codes or messages written precisely or displayed
- A description of the specific steps that immediately preceded the problem
- The severity code (ASG Support uses an escalated severity system to prioritize service to our clients. The severity codes and their meanings are listed below.)
- Verify whether you received an ASG Service Pack for this product. It may include information to help you resolve questions regarding installation of this ASG product. The Service Pack instructions are in a text file on the distribution media included with the Service Pack.

If You Receive a Voice Mail Message:

1. Follow the instructions to report a production-down or critical problem.
2. Leave a detailed message including your name and phone number. A Support representative will be paged and will return your call as soon as possible.
3. Please have the information described above ready for when you are contacted by the Support representative.

Severity Codes and Expected Support Response Times

Severity	Meaning	Expected Support Response Time
1	Production down, critical situation	Within 30 minutes
2	Major component of product disabled	Within 2 hours
3	Problem with the product, but customer has work-around solution	Within 4 hours
4	"How-to" questions and enhancement requests	Within 4 hours

ASG provides software products that run in a number of third-party vendor environments. Support for all non-ASG products is the responsibility of the respective vendor. In the event a vendor discontinues support for a hardware and/or software product, ASG cannot be held responsible for problems arising from the use of that unsupported version.

Business Hours Support

Your Location	Phone	Fax	E-mail
United States and Canada	800.354.3578	239.263.2883	support@asg.com
Australia	61.2.9460.0411	61.2.9460.0280	support.au@asg.com
England	44.1727.736305	44.1727.812018	support.uk@asg.com
France	33.141.028590	33.141.028589	support.fr@asg.com
Germany	49.89.45716.222	49.89.45716.400	support.de@asg.com
Singapore	65.6332.2922	65.6337.7228	support.sg@asg.com
All other countries:	1.239.435.2200		support@asg.com

Non-Business Hours - Emergency Support

Your Location	Phone	Your Location	Phone
United States and Canada	800.354.3578		
Asia	65.6332.2922	**Japan/Telecom**	0041.800.9932.5536
Australia	0011.800.9932.5536	**Netherlands**	00.800.3354.3578
Denmark	00.800.9932.5536	**New Zealand**	00.800.9932.5536
France	00.800.3354.3578	**Singapore**	001.800.3354.3578
Germany	00.800.3354.3578	**South Korea**	001.800.9932.5536
Hong Kong	001.800.9932.5536	**Sweden/Telia**	009.800.9932.5536
Ireland	00.800.9932.5536	**Switzerland**	00.800.9932.5536
Israel/Bezeq	014.800.9932.5536	**Thailand**	001.800.9932.5536
Japan/IDC	0061.800.9932.5536	**United Kingdom**	00.800.9932.5536
		All other countries	1.239.435.2200

ASG Web Site

Visit http://www.asg.com, ASG's World Wide Web site.

Submit all product and documentation suggestions to ASG's product management team at http://www.asg.com/asp/emailproductsuggestions.asp.

If you do not have access to the web, FAX your suggestions to product management at (239) 263-3692. Please include your name, company, work phone, e-mail ID, and the name of the ASG product you are using. For documentation suggestions include the publication number located on the publication's front cover.

THE MANAGER FAMILY

MANAGER FAMILY - MSP brings you a unique offering - the MANAGER Family of dictionary driven Program Products, winner of the ICP $250 Million Award. MANAGER Products and their dictionaries may be integrated to maximize the return on your software and on your dictionary investment.

USER CONFIGURED - Every MANAGER Product consists of a nucleus with optional additional facilities. You select the options you need, and MSP supplies your chosen configuration.

WORLD-WIDE SUPPORT - All MANAGER Products are backed by comprehensive support, documentation, education and user group activities on a world-wide basis.

METHODMANAGER™

Automating Information Engineering: the tailorable repository-driven Environment for ENABLING your AD/Cycle Strategy and for automating your Information Engineering Life Cycle

managerVIEW™

The Intelligent Workstation based graphical Information Engineering tool driven by the mainframe resident Corporate Dictionary/Repository

CONTROLMANAGER™

The dictionary driven End User Facility for the MANAGER Family, winner of the ICP $50 Million Award

DATAMANAGER™

The Corporate Dictionary/Repository driven Information Resource Management System, winner of the ICP $100 Million Award

DESIGNMANAGER™

The dictionary driven Interactive Data and Enterprise Modeling System for logical and physical database design

DICTIONARYMANAGER™

The Corporate Dictionary/Repository driven Interchange System supporting the exchange of definitions between multiple vendor dictionaries/ repositories

PROJECTMANAGER™

The dictionary driven Project Resource Management and Budgetary Control System

SOURCEMANAGER™

The dictionary driven Application Development System

TESTMANAGER™

The Program/Structure Testing and Test Data Generation System

THE MSP MANAGER FAMILY

- **designed for the future; implemented today**
- **making Information Engineering simple.**

MSP TRADEMARKS

MSP's trademarks are registered in one or more principal markets throughout the world.

The following are trademarks of Management Systems & Programming Limited:

- MSP
- MANAGER SOFTWARE PRODUCTS
- CONTROLMANAGER
- DATAMANAGER
- DESIGNMANAGER
- DICTIONARYMANAGER
- METHODMANAGER
- PROJECTMANAGER
- SOURCEMANAGER
- TESTMANAGER
- TEXTMANAGER
- InfoBank
- InfoDictionary
- InfoSystem
- InfoView
- CADIE - Computer Aided Data and Information Engineering.
- MIDLINE
- MIDWEEK

The following is a trademark of WHDL - MSP INC:

- managerVIEW.

TRADEMARKS ACKNOWLEDGEMENTS

MSP acknowledges that certain proprietary products or services or programs mentioned within this publication or within other MSP publications are distributed under Trademarks or Registered Trademarks of the vendors who own and/or distribute the products or services or programs in the country in question.

ABOUT THIS PUBLICATION

This publication is one of a series describing the MANAGER Family of Program Products developed by MSP for organizations seeking to automate and manage their application development and maintenance effort.

It describes the METHODMANAGER Dictionary/Repository Information Model (MDRIM) as supplied by MSP: the types of entities and relationships that may be documented in your corporate repository.

WHO SHOULD READ THIS PUBLICATION?

This publication is intended for all METHODMANAGER repository users.

Details of how the MSP-supplied MDRIM may be tailored to your installation's own requirements can be found in the METHODMANAGER Administration manual (MMR-ADMIN).

HOW THIS PUBLICATION IS ORGANIZED

This publication is arranged as follows:

Chapter 1 The Dictionary/Repository Information Model
An introduction to the Dictionary/Repository Information Model.

Chapter 2 Member Type Descriptions
This chapter describes the member types supplied to enable you to model your organization's information resource in the repository.

Appendix 1 Enterprise Entity Relationship Data Submodel
This chapter describes how you can create a business model of the data requirements of your organization.

Appendix 2 Common Clauses and Attributes
This chapter describes common clauses and attributes available to document entities and relationships in the repository.

OTHER MSP PUBLICATIONS

The following publications are cited in this publication:

METHODMANAGER Administration: (MMR-ADMIN)

A range of publications is available covering the MANAGER Family of Program Products. Details are published in the MANAGER Products Docuware Bulletin, which is distributed to all users of MANAGER Products.

CONVENTIONS

The following conventions apply to syntax diagrams that appear in this manual.

Diagrams are read from left to right along a continuous line (the "main path"). Keywords and variables appear on, above, or below the main path.

►► at the beginning of a line indicates the start of a statement.

►◄ at the end of a line indicates the end of a statement.

—► at the end of a line indicates that the statement continues on the line below.

►— at the beginning of a line indicates that the statement is continued from the line above.

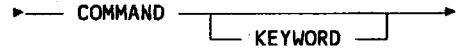
Keywords are in upper-case characters. Keywords and any required punctuation characters or symbols are highlighted. Permitted truncations are not indicated.

Variables are in lower-case characters.

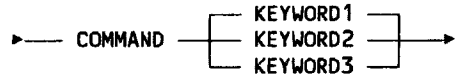
Statement identifiers appear on the main path of the diagram:

A **required keyword** appears on the main path:

An **optional keyword** appears below the main path:

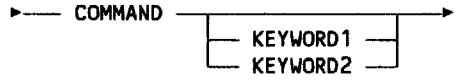


Where there is a **choice of required keywords,** the keywords appear in a vertical list; one of them is on the main path:

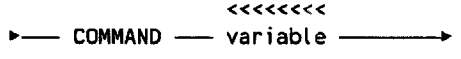


or

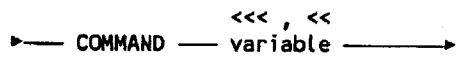
Where there is a **choice of optional keywords,** the keywords appear in a vertical list, below the main path:



The **repeat** symbol, <<<<<<, above a keyword or variable, or above a whole clause, indicates that the keyword, variable, or clause may be specified more than once:

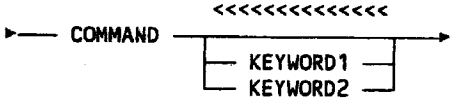


A repeat symbol broken by a comma indicates that if the keyword, variable or clause is specified more than once, a comma must separate each instance of the keyword, variable or clause:



The **repeat symbol above a list of keywords (one of which appears on the main path)** indicates that any one or more of the keywords may be specified; one at least **must** be specified:

The **repeat symbol above a list of keywords (all of which are below the main path)** indicates that any one or more of the keywords may be specified, but they are all **optional** :



CONTENTS

	ABOUT THIS PUBLICATION	vii
	LIST OF FIGURES	xiii
1	THE DICTIONARY/REPOSITORY INFORMATION MODEL	1-1
1.1	INTRODUCTION	1-1
1.2	MEMBER TYPES SUPPLIED WITH MDRIM	1-2
2	MEMBER TYPE DESCRIPTIONS	2-1
APP.1	ENTERPRISE ENTITY-RELATIONSHIP DATA SUBMODEL	App.1-1
APP.1.1	INTRODUCTION	App.1-1
APP.1.2	BUSINESS MODELS	App.1-1
App.1.2.1	Introduction	App.1-1
App.1.2.2	Properties of Business Relationships	App.1-5
App.1.2.3	Properties of Has-Attributes Relationships	App.1-6
APP.1.3	THE MEMBER TYPES	App.1-7
APP.1.4	TYPICAL BUSINESS CONSTRUCTS	App.1-8
App.1.4.1	Introduction	App.1-8
App.1.4.2	Unattributed Entity	App.1-10
App.1.4.3	Attributed Entity	App.1-12
App.1.4.4	Binary Business Relationship	App.1-14
App.1.4.5	Attributed Business Relationship	App.1-17
App.1.4.6	Business Relationship on a Business Relationship	App.1-19
App.1.4.7	Existential Dependency	App.1-21

APP.2 COMMON CLAUSES AND ATTRIBUTES App.2-1
APP.2.1 INTRODUCTION App.2-1
APP.2.2 COMMON CLAUSE DEFINITIONS App.2-2
APP.2.3 COMMON ATTRIBUTES App.2-9
APP.2.4 IEW COMMON ATTRIBUTES App.2-10

AMENDMENT RECORD AR-1

USER'S COMMENTS FORM

LIST OF FIGURES

Figure App.1.1	Model of an Enterprise	App.1-4
Figure App.1.2	Diagram of the RIM Submodel	App.1-10
Figure App.1.3	Unattributed Entity	App.1-11
Figure App.1.4	Representation of Unattributed Entity	App.1-11
Figure App.1.5	Attributed Entity	App.1-13
Figure App.1.6	Representation of Attributed Entity	App.1-14
Figure App.1.7	Binary Business Relationship	App.1-16
Figure App.1.8	Representation of Binary Business Relationship	App.1-16
Figure App.1.9	Attributed Business Relationship	App.1-17
Figure App.1.10	Representation of Attributed Business Relationship	App.1-18
Figure App.1.11	Business Relationship on a Business Relationship	App.1-20
Figure App.1.12	Representation of Business Relationship on a Business Relationship	App.1-21
Figure App.1.13	Weak Entity	App.1-24
Figure App.1.14	Representation of Weak Entity	App.1-24
Figure App.1.15	N-ary Relationship	App.1-25
Figure App.1.16	Representation of N-ary Relationship	App.1-25

1 THE DICTIONARY/REPOSITORY INFORMATION MODEL

1.1 INTRODUCTION

A repository information model (RIM) defines the types of entities, and the relationships between entities, that can be documented in a repository: it defines the structure of information in the repository and forms the repository's schema. MSP supply you with a fully defined RIM - the METHODMANAGER Dictionary/Repository Information Model (MDRIM) - for use in your corporate repository. We supply the MDRIM as UDS table DU016, for direct loading onto your MP-AID.

The MDRIM provides you with a comprehensive range of member types that support the documentation of cross life cycle information. The MDRIM is designed to support the definition, in the repository, of your organization's entire information resource, including business and data processing entities: it enables you to model the organization's functions and data that your installation relies on. The MDRIM consists of sub-models designed to support the definition of entities and relationships relating to different activities, such as project management, or Database Management Systems (DBMSs) such as DB2 and IMS.

You can create your own repository information models or adapt the supplied MDRIM to suit your organization's needs: the repository definitions that define the MDRIM are held in your Administration Repository, which requires its own RIM, supplied as UDS table DU777. The HIERARCHY member that defines the MDRIM, UH016, may be tailored to your installation's requirements in your Administration Repository.

For details on how you can tailor MSP-supplied RIMs and on how to create new RIMs, refer to your METHODMANAGER Administration manual (MMR-ADMIN).

The MDRIM may also be used in non-METHODMANAGER repositories supplied by MSP.

1.2 MEMBER TYPES SUPPLIED WITH MDRIM

The predefined member types provided by MANAGER Products are as follows:

- for strategic information planning:

- CRITICAL-SUCCESS-FACTOR
- DATAFLOW
- ENTITY
- EXTERNAL
- FILE
- FORM
- FUNCTION
- GOAL
- GROUP
- ITEM
- LOCATION
- ORGANIZATIONAL-UNIT
- REPORT
- REQUIREMENT
- SCREEN
- SUBJECT-AREA
- SYSTEM
- VIEWSET

- for application development:

- ACCESSVIEW
- BUSINESS-RELATIONSHIP
- CONCEPTUAL-RECORD
- CONCEPTUAL-RELATION
- DATAFLOW
- DATASTORE
- DIALOG
- ENTITY
- EXTERNAL
- FILE
- FORM
- FUNCTION
- GROUP
- GOAL
- HAS-ATTRIBUTES
- IEW member types
- ITEM
- LOCATION
- MAP
- MODULE
- ORGANIZATIONAL-UNIT
- PROCESS-VIEW
- PROCESSING-RULE
- PROGRAM
- REPORT
- REQUIREMENT
- REUSABLE-CODE
- SCREEN
- SYSTEM
- TRANSACTION
- USERVIEW
- VIEWSET
- The Database Management System Interfaces (DB2, SQL/DS, IMS/DL1, IDMS, ADABAS, SUDS, SYSTEM-2000/80, TOTAL and MARK IV Interface member types

- for producing hard-copy documentation using the supplied Documentation functions:

  - DOCUMENT

- for project management

  - PROJECT
  - TASK

- for life cycle management

  - ACTIVITY
  - LIFE-CYCLE
  - LIFE-CYCLE-OBJECT-TYPE
  - PHASE

2 MEMBER TYPE DESCRIPTIONS

This chapter describes the member types supplied by MANAGER Products. Member types are documented in alphabetical order of member type name.

For each member type, details of the clauses and more commonly used relationships permitted between member types are given, followed by the syntax.

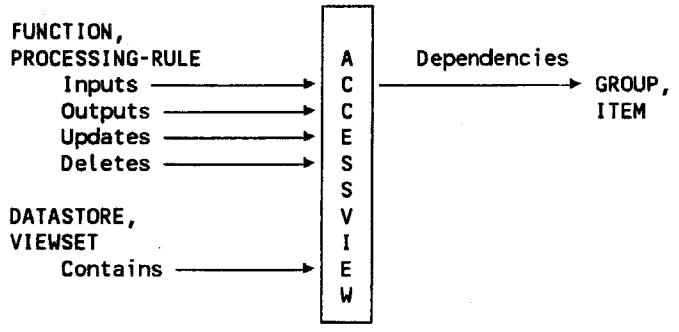
Note:

- to display all the relationships permitted between member types, use the SHOW UDS command. For details, refer to the *Dictionary/Repository User's Guide* (MPR-DRUG)
- for ADW member types, see **IEW**
- for IMS member types, see **DL/1.**

ACCESSVIEW

An ACCESSVIEW is a set of dependencies which express an elementary function's requirements for accessing the values of data elements (items and certain groups) in the entity model.

The more commonly used relationships to and from ACCESSVIEW are:



DEPENDENCIES
Contains the names of data elements in the LHS of a dependency, type of dependency (FD or MVD), the number of elements if it is an MVD, and the name of each data element in the RHS of the dependency.

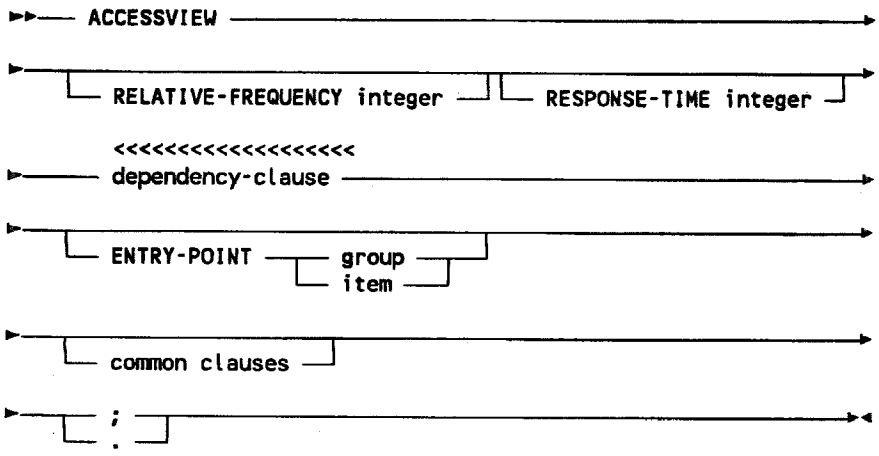
RELATIVE-FREQUENCY
Stores the relative frequency of use of this accessview. The relative frequency with which data is accessed determines how it should be stored for optimum response.

RESPONSE-TIME
Represents the relative response-time desired for the accessview. The relative response-time needed depends on how often this particular accessview is going to be used.

ENTRY-POINT
Identifies the GROUP or ITEM which users of this particular accessview must select to begin accessing the information held on the system.

ACCESSVIEW

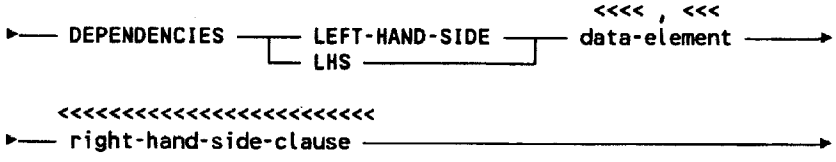
Syntax



where

integer is an integer value of up to 18 digits, optionally preceded by a sign

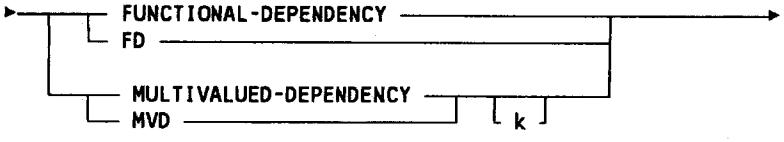
dependency-clause is:

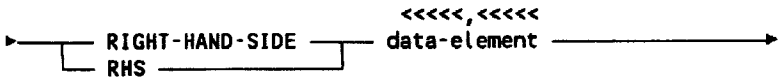


where

data-element is the name of a data element that is held in the modeling repository as an ITEM or GROUP member or is to be entered in the modeling repository as a dummy ITEM

right-hand-side-clause is:





where

k is an unsigned integer indicating the multiplicity for a multivalued dependency, that is, the average number of values (or sets of values) of the right-hand side of the dependency determined by a given value of the left-hand side. If not specified for a dependency, the default value of k is taken to be 1. (The multiplicity of a functional dependency is automatically taken to be 1.)

data-element is as defined above

group is the name of a GROUP member

item is the name of an ITEM member

common clauses are any of the clauses common to all member types

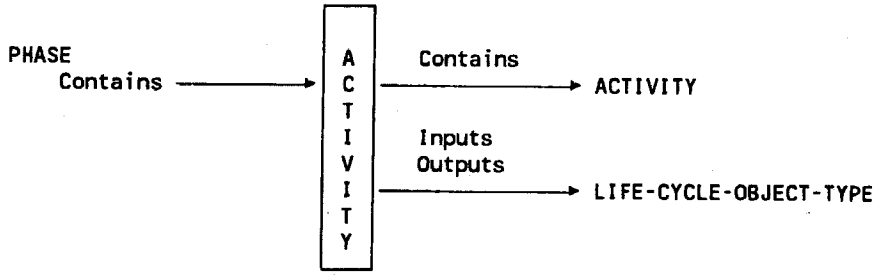
Note: the commas and delimiters shown in the above syntax are required when defining an ACCESSVIEW member via the command interface.

Refer to Appendix 2 for details of the common clauses.

ACTIVITY

An ACTIVITY describes a stage within a phase which leads to a defined (intermediate) result in the form of one or several LIFE CYCLE OBJECT TYPEs. For an activity to be executed, prerequisites in the form of input life cycle object types must be present.

The more commonly used relationships to and from ACTIVITY are:



INPUTS
Prerequisite results

OUTPUTS
Life cycle object type produced by activity.

CONTAINS
Must contain the name of every subactivity which is a part of this activity.

OPTION
Short description of project management components, as shown on the user interface.

OPTION-TEXT
Long description of project management components, as shown on the user interface.

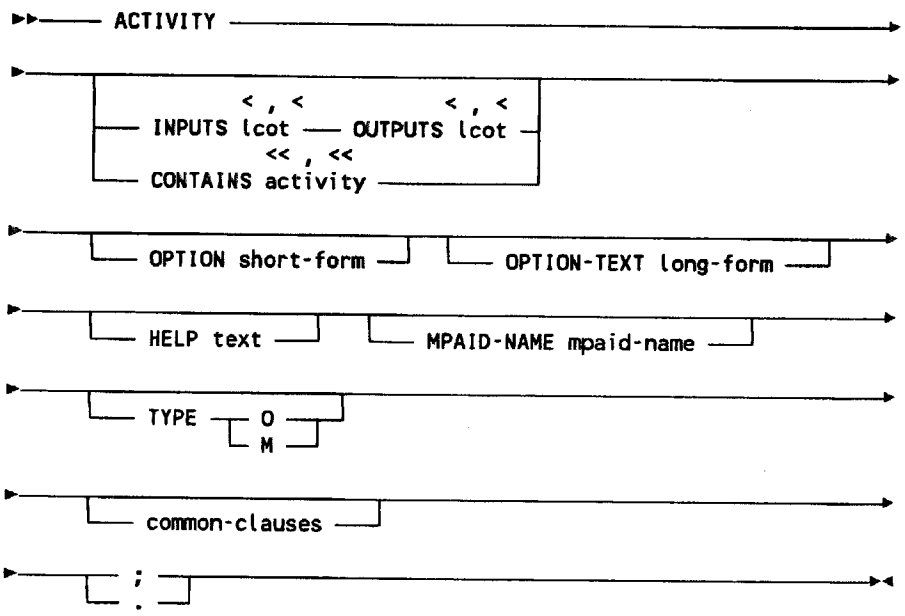
HELP
Description of the member with no limit on length.

TYPE
Indicates whether or not the result must be present/created.

MPAID-NAME
MPAID-NAME for the generation of the life cycle.

ACTIVITY

Syntax



where

lcot is the name of a LIFE-CYCLE-OBJECT-TYPE member

activity is the name of an ACTIVITY member

short-form is a delimited string of up to 2 characters

long-form is a delimited string of up to 50 characters

text is a maximum of 32767 delimited character strings, each containing a maximum of 256 characters

mpaid-name is a name of up to 10 characters

common-clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common-clauses.

ADABAS

For details of the ADABAS interface member types:

- ADABAS-DATABASE
- ADABAS-FILE

please refer to the ADABAS Interface manual (DMR-ADA).

BUSINESS-RELATIONSHIP

You use a BUSINESS-RELATIONSHIP member to define a business relationship.

For details of business relationships refer to the METHODMANAGER Administration manual (MMR-ADMIN).

SOURCE
To specify the source member, enter:

```
SOURCE name
```

name is the name of an ENTITY member or a BUSINESS-RELATIONSHIP member.

TARGET
To specify the target member, enter:

```
TARGET name
```

name is defined above.

FORWARD-VERB
To specify the forward name, enter:

```
FORWARD-VERB name
```

name is a string of up to 32 characters.

INVERSE-VERB
To specify the inverse name, enter:

```
INVERSE-VERB name
```

name is defined above.

LONG-FORWARD-NAME
To specify the long forward name, enter:

```
LONG-FORWARD-NAME name
```

name is a string of up to 80 characters.

LONG-INVERSE-NAME
To specify the long inverse name, enter:

```
LONG-INVERSE-NAME name
```

name is defined above.

SOURCE-MANDATORY
To specify that the source is mandatory, enter:

```
SOURCE-MANDATORY M
```

To specify that the source is optional, enter:

```
SOURCE-MANDATORY O
```

If the SOURCE-MANDATORY clause is omitted, optional is assumed.

TARGET-MANDATORY
To specify that the target is mandatory, enter:

```
TARGET-MANDATORY M
```

To specify that the target is optional, enter:

```
TARGET-MANDATORY O
```

If the TARGET-MANDATORY clause is omitted, optional is assumed.

SOURCE-MINIMUM-CARDINALITY and
SOURCE-MAXIMUM-CARDINALITY
To specify a source cardinality of p,q, enter:

```
SOURCE-MINIMUM-CARDINALITY p
SOURCE-MAXIMUM-CARDINALITY 'q'
```

p and **q** are positive integers and $q >= p > 0$.

To specify a source cardinality of many, an unrestricted source cardinality, enter:

```
SOURCE-MAXIMUM-CARDINALITY 'm'
```

Note: m is a literal not a variable.

To specify a source cardinality of p,many, enter:

```
SOURCE-MINIMUM-CARDINALITY p
```

p is a positive integer.

If both clauses are omitted, a source cardinality of many is assumed.

To specify a target cardinality of p,q, enter:

```
TARGET-MINIMUM-CARDINALITY p
TARGET-MAXIMUM-CARDINALITY 'q'
```

p and **q** are positive integers and q > = p.

To specify a target cardinality of many, an unrestricted target cardinality, enter:

```
TARGET-MAXIMUM-CARDINALITY 'm'
```

Note: m is a literal not a variable.

To specify a target cardinality of p,many, enter:

```
TARGET-MINIMUM-CARDINALITY p
```

p is a positive integer.

If both clauses are omitted, a target cardinality of many is assumed.

Note that clauses specifying a maximum cardinality are text clauses so that the non-specific m for many can be used.

SOURCE-DEPENDENT
To specify that the source is dependent, enter:

```
SOURCE-DEPENDENT D
```

To specify that the source is independent, enter:

```
SOURCE-DEPENDENT I
```

If the SOURCE-DEPENDENT clause is omitted, independent is assumed.

TARGET-DEPENDENT
To specify that the target is dependent, enter:

```
TARGET-DEPENDENT D
```

To specify that the target is independent, enter:

```
TARGET-DEPENDENT I
```

If the TARGET-DEPENDENT clause is omitted, independent is assumed.

SOURCE-CONTROLLED
To specify that the source is controlled, enter:

```
SOURCE-CONTROLLED C
```

To specify that the source is not controlled, enter:

```
SOURCE-CONTROLLED U
```

If the SOURCE-CONTROLLED clause is omitted, not controlled (that is, uncontrolled) is assumed.

TARGET-CONTROLLED
To specify that the target is controlled, enter:

```
TARGET-CONTROLLED C
```

To specify that the target is not controlled, enter:

```
TARGET-CONTROLLED U
```

If the TARGET-CONTROLLED clause is omitted, not controlled (that is, uncontrolled) is assumed.

MAXIMUM-OCCURRENCE
To specify the maximum number of occurrences of the relationship, enter:

```
MAXIMUM-OCCURRENCE integer
```

integer is a positive integer.

MINIMUM-OCCURRENCE
To specify the minimum number of occurrences of the relationship, enter:

```
MINIMUM-OCCURRENCE integer
```

integer is defined above.

OCCURRENCE
To specify the expected number of occurrences of the relationship, enter:

```
OCCURRENCE integer
```

integer is a positive integer within the OCCURRENCE range.

To specify the growth rate period, enter:

```
GROWTH-RATE-PERIOD 'string'
```

string is one of the following:

```
MONTH
DAY
YEAR
QUARTER
etc.
```

To specify the growth rate percentage, enter:

```
GROWTH-RATE-PERCENT integer
```

integer is any integer.

To specify a median source cardinality, enter:

```
SOURCE-MEDIAN-CARDINALITY int
```

int is a positive integer within the source cardinality range.

To specify a median target cardinality, enter:

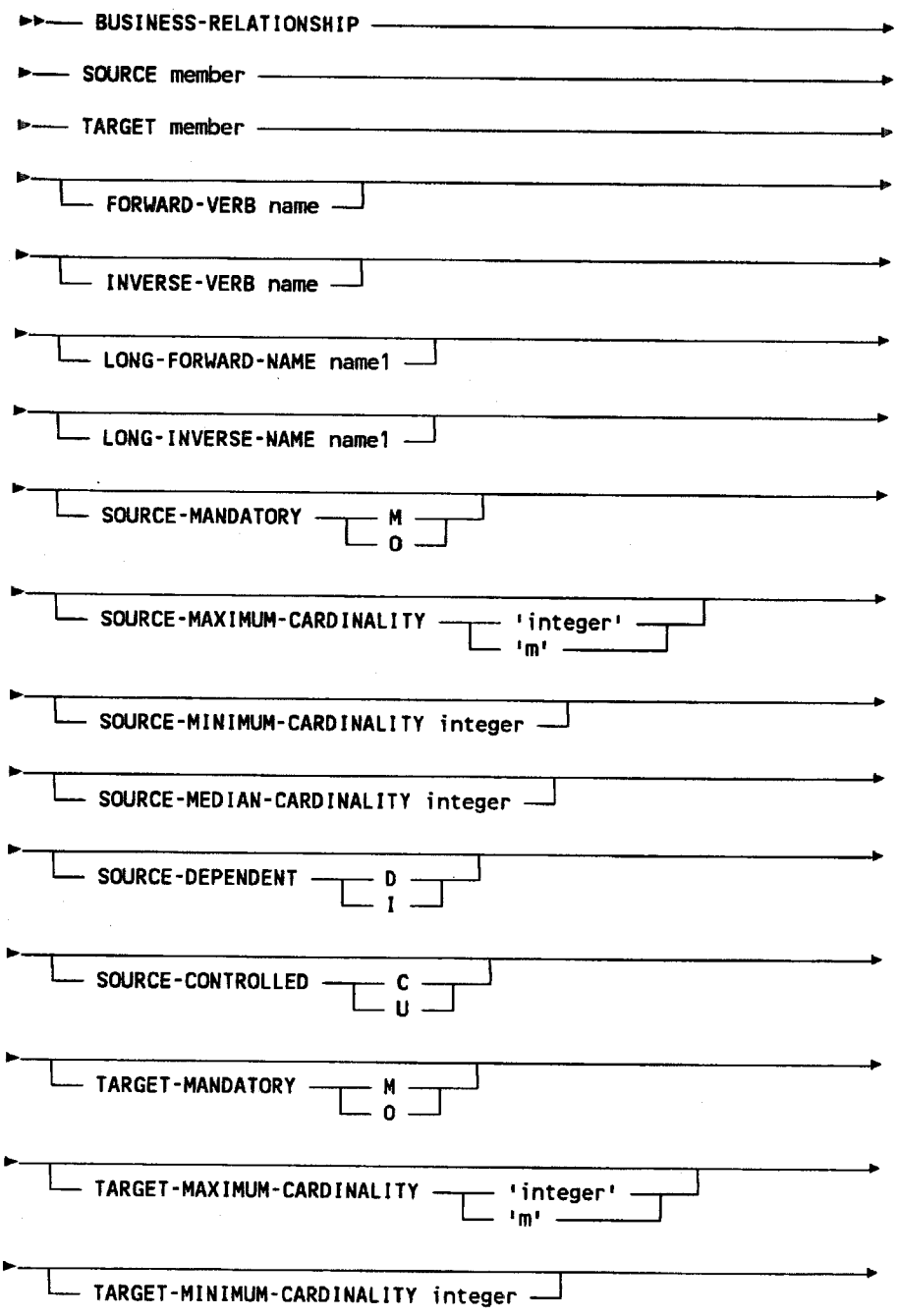
```
TARGET-MEDIAN-CARDINALITY int
```

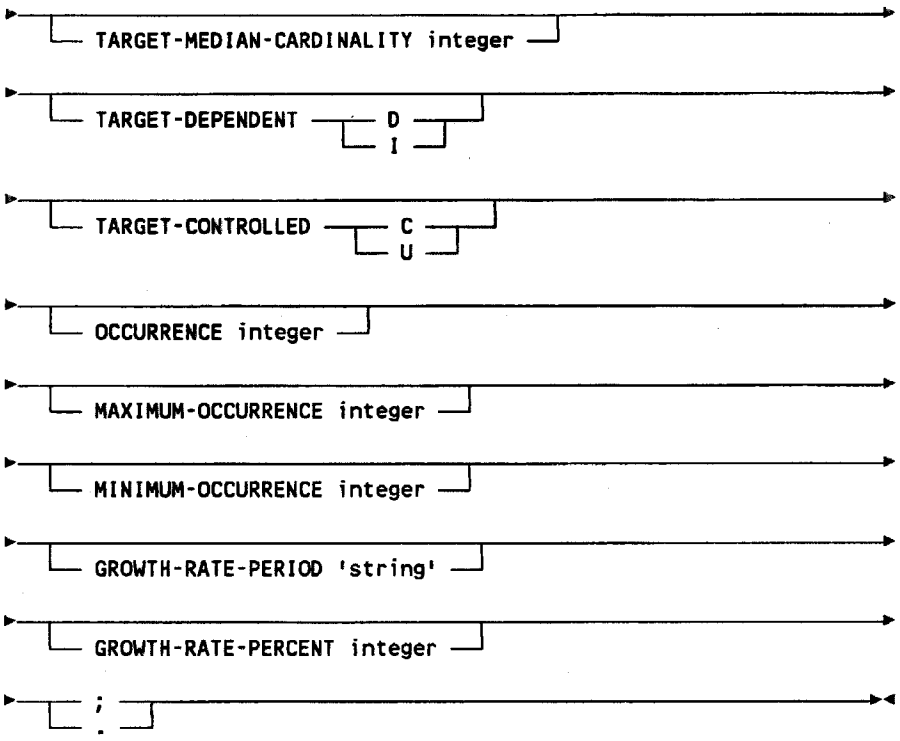
int is a positive integer within the target cardinality range.

All these clauses are optional and have no default values.

BUSINESS-RELATIONSHIP

Syntax





where

member is a repository member name

name is a string of up to 32 characters

name1 is a string of up to 80 characters

integer is an integer

Note: m is a literal, not a variable

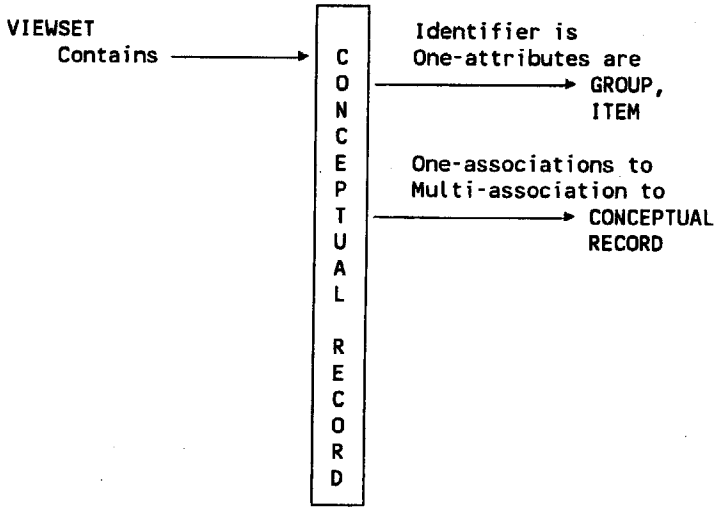
string is a string of up to 32 characters.

Note: the delimiters shown in the above syntax are required when defining a BUSINESS-RELATIONSHIP member via the command interface.

CONCEPTUAL-RECORD

A CONCEPTUAL-RECORD consists of a primary key and its descriptive elements. It is the result of the normalization process, can never consist solely of the primary key but will have associations to other conceptual records.

The more commonly used relationships to and from CONCEPTUAL-RECORD are:



IDENTIFIER IS
The data elements specified form the primary key of the relation documented in this record. The primary key is the key which satisfies users' primary access requirements.

ONE-ATTRIBUTES ARE
May contain the name of one or more data elements.
Elements named in this clause occur only once in the member.

ONE-ASSOCIATIONS
Lists the members with which this member has a one-association. All these members also have some association with each other. The one-association can be labelled to indicate the action of this member on the members listed in the clause.

MULTI-ASSOCIATIONS
This clause may contain the name of one or more conceptual records.

A multi-association occurs when a conceptual record has an association with more than one occurrence of another conceptual record. This clause names the other conceptual records with which this member has a multi-association.
The multi-association may be given a label by which it is identified in the ASSOCIATION-DETAILS clause.
For example:
For a conceptual record whose name is:

```
CR-EMPLOYEE
```

you can specify the clause:

```
MULTI-ASSOCIATION LABEL REPORT TO EN-DEPT
```

which means that the employee reports to more than one department.

ASSOCIATION-DETAILS
Contains a full description of each association named in the ONE-ASSOCIATIONS TO and MULTI-ASSOCIATION TO clauses. The names of these associations are recorded in their LABEL subclauses.

SUB-ENTITIES ARE
Lists the sub-entities which comprise this member. All sub-entities of this member should be named in this clause.

MINIMUM-OCCURRENCE
To specify the minimum number of occurrences of the relationship, enter:

```
MINIMUM-OCCURRENCE integer
```

integer is a positive integer.

CONCEPTUAL-RECORD

OCCURRENCE

To specify the expected number of occurrences of the relationship, enter:

```
OCCURRENCE integer
```

integer is a positive integer within the OCCURRENCE range.

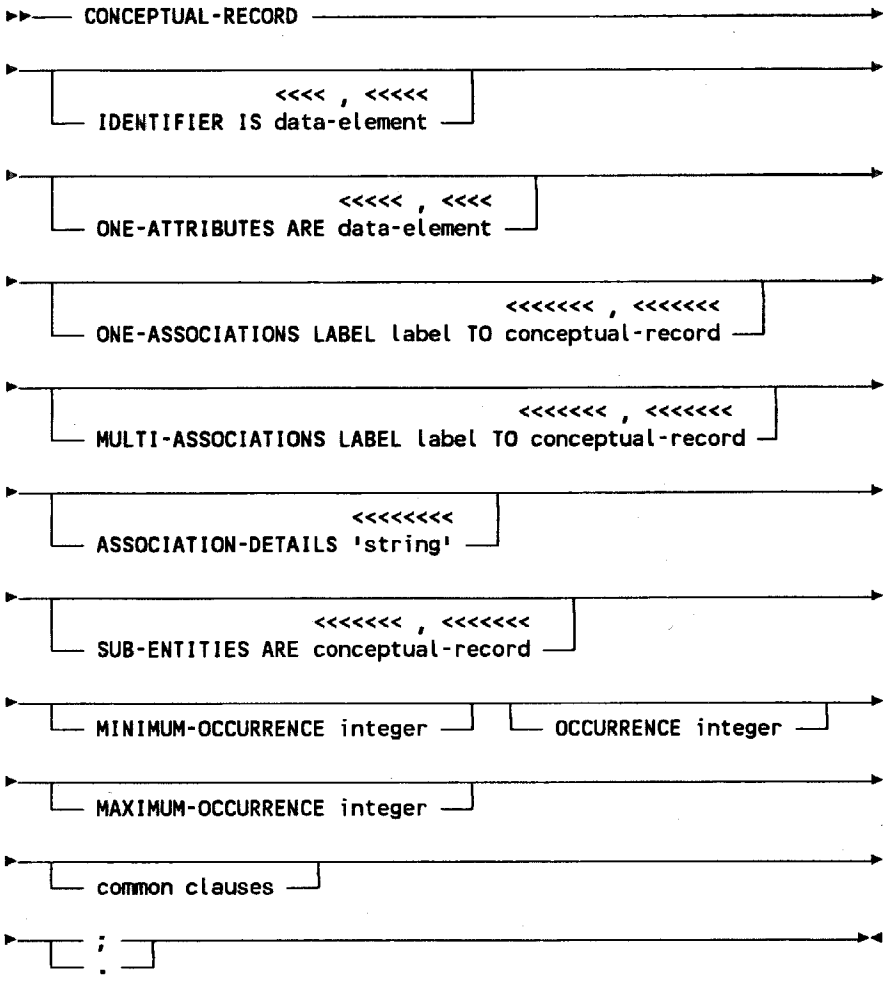
MAXIMUM-OCCURRENCE

To specify the maximum number of occurrences of the relationship, enter:

```
MAXIMUM-OCCURRENCE integer
```

integer is a positive integer.

Syntax



where

data-element is the name of a GROUP or ITEM member

label is a text string of up to 32 characters

conceptual-record is the name of a CONCEPTUAL-RECORD member

'string' is a string of up to 32 characters

CONCEPTUAL-RECORD

integer is an integer value of up to 18 digits, optionally preceded by a sign

common clauses are any of the clauses common to all member types.

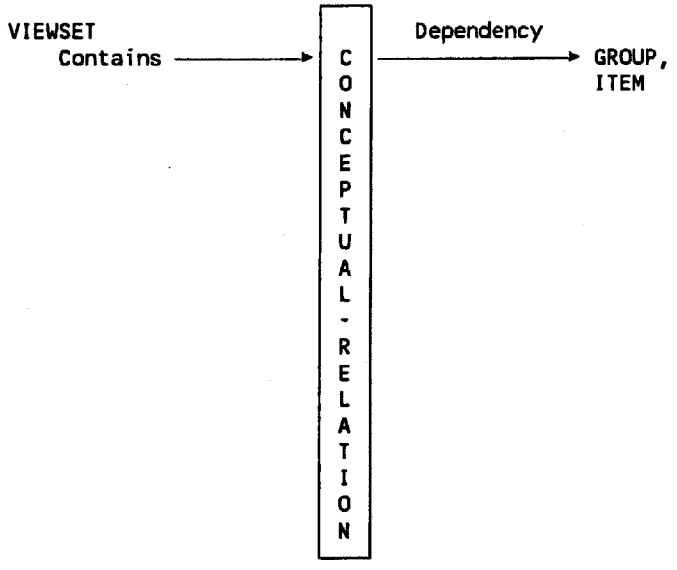
Note: the delimiters shown in the above syntax are required when defining a CONCEPTUAL-RECORD member via the command interface.

Refer to Appendix 2 for details of the common clauses.

CONCEPTUAL-RELATION

A CONCEPTUAL-RELATION is a logical description of a set of data elements in at least first normal form. It is a result of the normalization process and could consist solely of a primary key but cannot have associations to other conceptual relations.

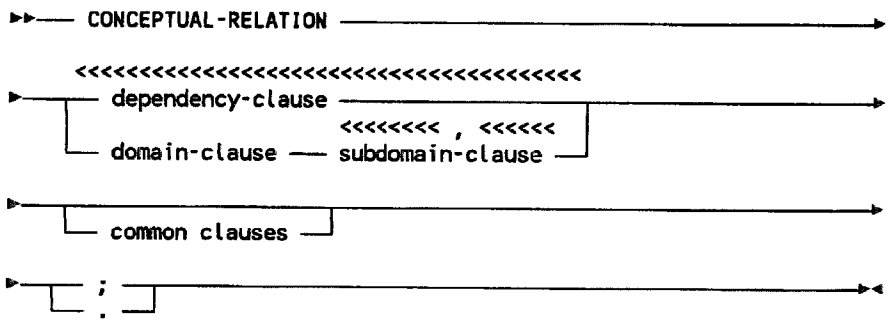
The more commonly used relationships to and from CONCEPTUAL-RELATION are:



DEPENDENCIES
Contains the names of data elements in the LHS of a dependency, type of dependency (FD or MVD), the number of elements if it is an MVD, and the name of each data element in the RHS of the dependency.

CONCEPTUAL-RELATION

Syntax

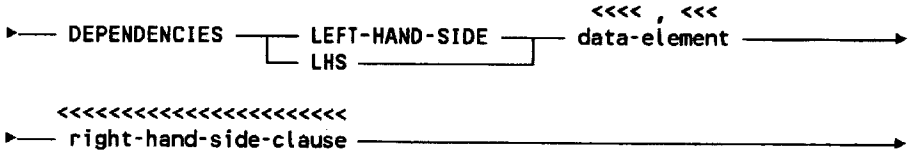


where

group is the name of a GROUP member

item is the name of an ITEM member

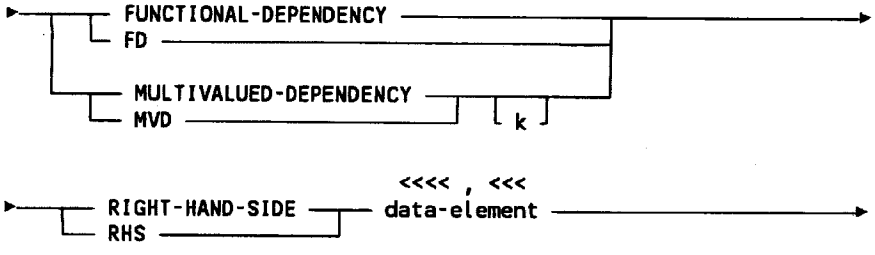
dependency-clause is:



where

data-element is the name of a data element that is held in the modeling repository as an ITEM or GROUP member or is to be entered in the modeling repository as a dummy ITEM

right-hand-side-clause is:

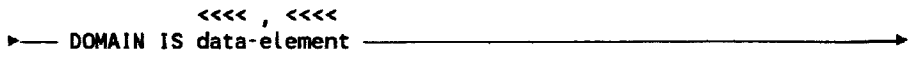


where

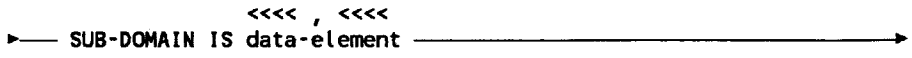
k is an unsigned integer indicating the multiplicity for a multivalued dependency, that is, the average number of values (or sets of values) of the right-hand side of the dependency determined by a given value of the left-hand side. If not specified for a dependency, the default value of k is taken to be 1. (The multiplicity of a functional dependency is automatically taken to be 1.)

data-element is as defined above

domain-clause is:



subdomain-clause is:



where **data-element** is as defined above

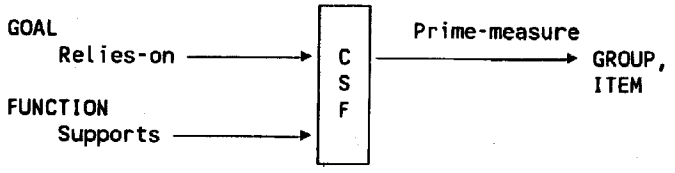
common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

CRITICAL-SUCCESS-FACTOR

A CRITICAL-SUCCESS-FACTOR (CSF) is one of the limited number of areas of business activity whose results, if they are satisfactory, will ensure successful competitive performance for the organizational unit and the attainment of its goals.

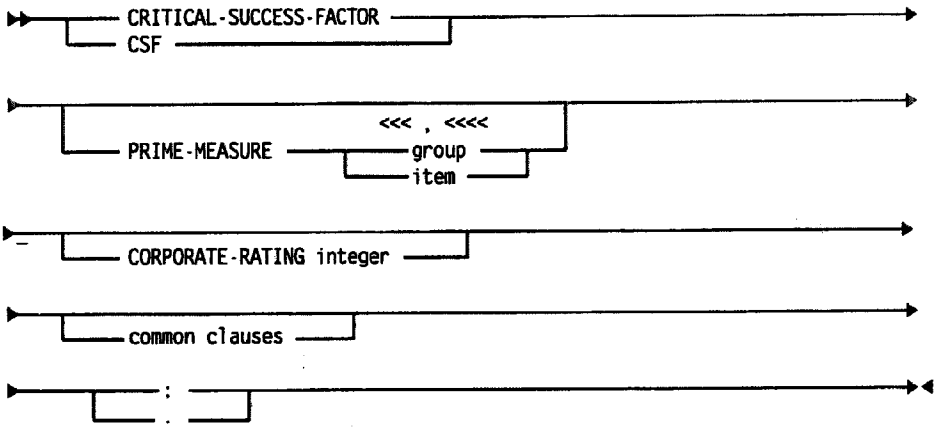
The more commonly used relationships to and from CRITICAL-SUCCESS-FACTOR are:



PRIME-MEASURE
Defines the prime measure of this Critical Success Factor (CSF). The prime measure is the set of data elements (ITEMs or GROUPs) whose values indicate the success or otherwise of this CSF.

CORPORATE-RATING
Stores an integer reflecting the importance that the enterprise attaches to this Critical Success Factor (CSF). Any scale of values can be used, but it must be used for every CSF in the enterprise.

Syntax



where

group is the name of a GROUP member

item is the name of an ITEM member

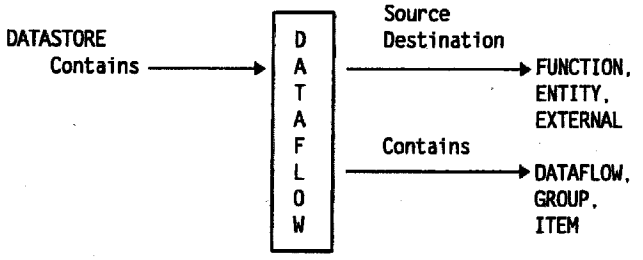
integer is an integer value of up to 18 digits, optionally preceded by a sign.

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DATAFLOW

A DATAFLOW is a collection of several data elements which are jointly transmitted from a source to a destination. One or both of the source and destination must be a function.
The more commonly used relationships to and from DATAFLOW are:



SOURCE
Contains the name of the member which is the source of the dataflow. It should only contain the name of one member, since a dataflow can logically have only one source. Optionally the condition necessary for the dataflow to occur can be specified by entering the condition name within the OPTIONAL UNDER-CONDITION subclause.

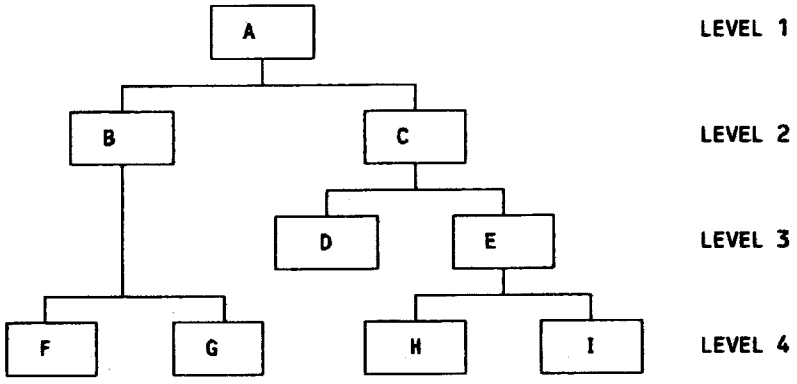
DESTINATION
Contains the name of the member which is the destination of the dataflow. It should only contain the name of one member, since a dataflow can logically have only one destination.

ENTITY-STATES
Lists each old-state (the entity's state before a dataflow) and its corresponding new-state (the entity's state after a dataflow).

CONTAINS
Contains the names of one or more DATAFLOW, GROUP and ITEM members, which make up the dataflow.

CONDITION-DETAILS
May contain further explanation of the condition described in the UNDER-CONDITION clause of the DATAFLOW member.

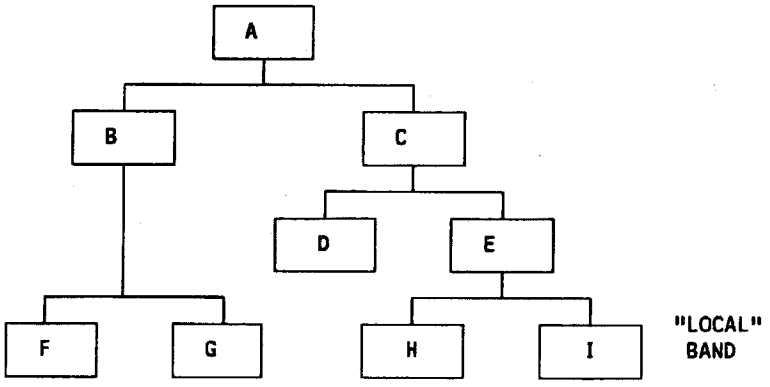
LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy. The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

BAND
The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.

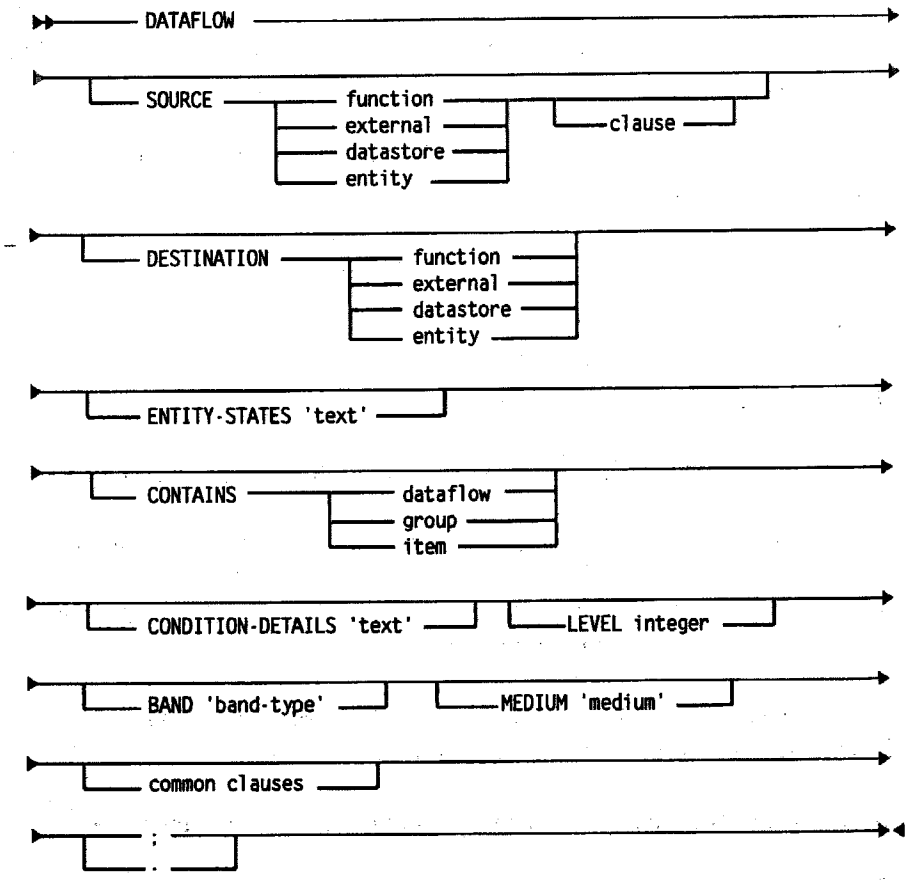
DATAFLOW



The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

MEDIUM
Specifies the medium by which a dataflow travels from source to destination, for example, telephone, letter, tape.

Syntax



where

function is the name of a FUNCTION member

external is the name of an EXTERNAL member

datastore is the name of a DATASTORE member

entity is the name of an ENTITY member

DATAFLOW

clause is:

where **condition-expression** is a string of up to 79 characters, including delimiters

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 256 characters

dataflow is the name of a DATAFLOW member

group is the name of a GROUP member

item is the name of an ITEM member

integer is an integer value of up to 18 digits, optionally preceded by a sign.

'band-type' is a text string of up to 78 characters, including delimiters.

'medium' may contain a text string of up to 32767 delimited character strings, each up to 252 characters long

Note: the delimiters shown in the above syntax are required when defining a DATAFLOW member via the command interface.

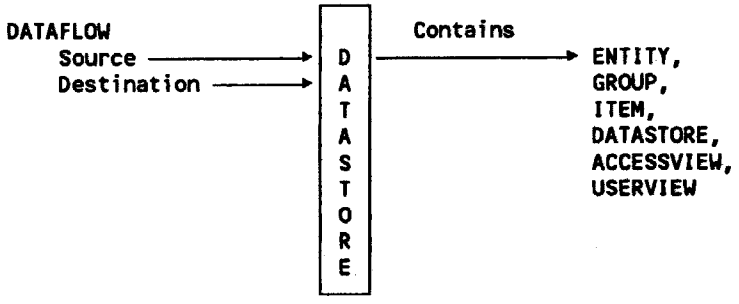
common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DATASTORE

A DATASTORE is a logical storage place for data consisting of related entities, groups, items and possibly other datastores.

The more commonly used relationships to and from DATASTORE are:



CONTAINS
This clause may contain the names of one or more groups or items or entities or subsidiary datastores.
For example:
For a DATASTORE member whose name is:

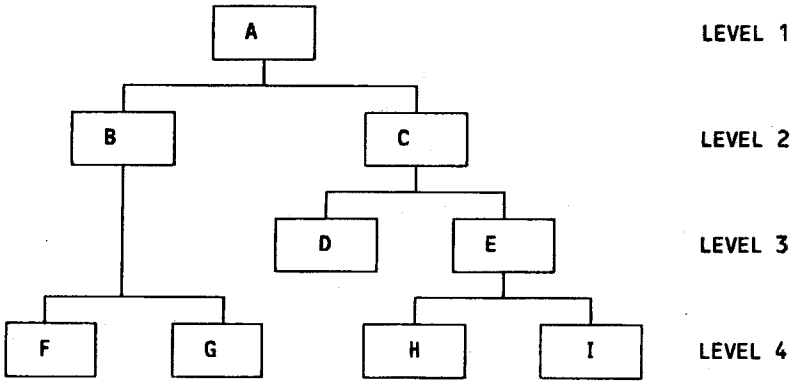
```
DS-CUSTOMER-ORDER
```

you could specify the clause:

```
CONTAINS
      EN-CUSTOMER
      EN-CUSTOMER-ORDER
```

LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy.
The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.

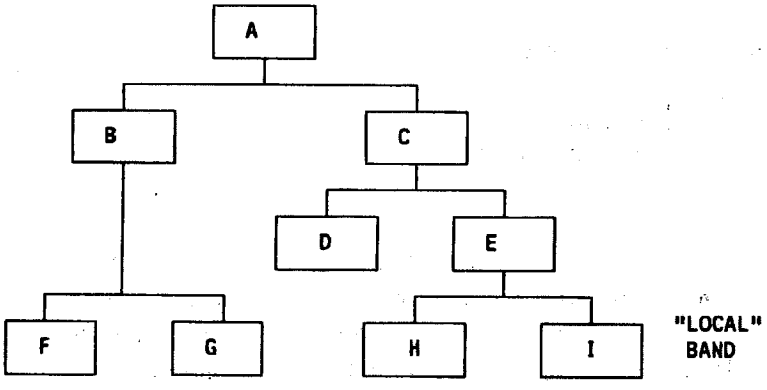
DATASTORE



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

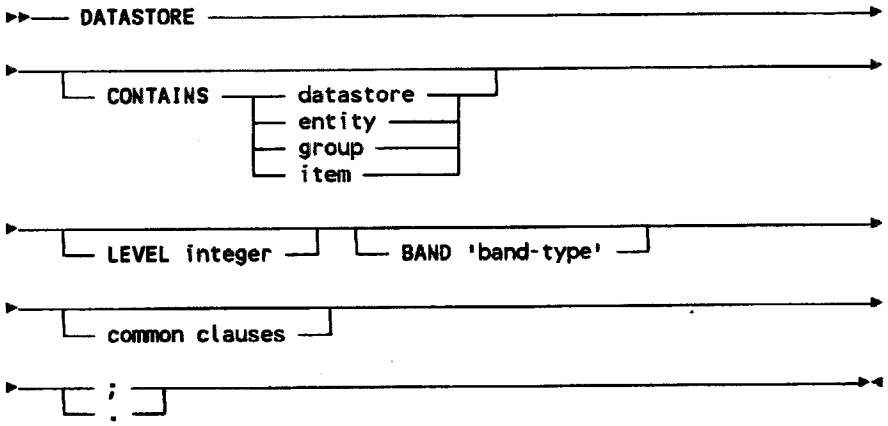
BAND

The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.



The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

Syntax



where

datastore is the name of a DATASTORE member

entity is the name of an ENTITY member

group is the name of a GROUP member

item is the name of an ITEM member

integer is an integer value of up to 18 digits, optionally preceded by a sign.

'band-type' is a text string of up to 78 characters, including delimiters

Note: the delimiters shown in the above syntax are required when defining a DATASTORE member via the command interface.

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DB2-ALIAS

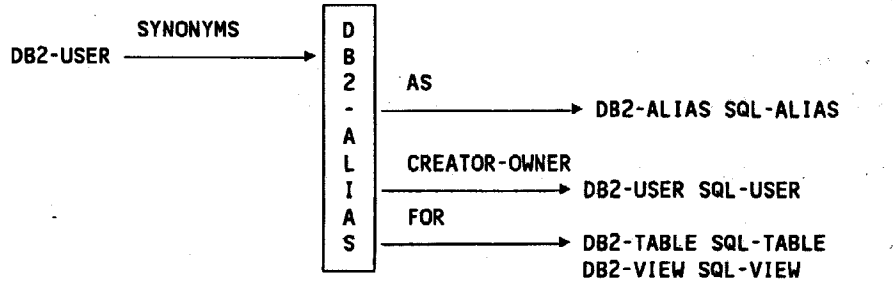
In the DB2 environment, you can create a local DB2 alias for a table or view at a remote location in a distributed network, if you have access privileges to the remote table or view. Local users who do not know the location of the table or view can then access it. DB2 aliases are represented on the repository as DB2-ALIAS members.

All the clauses available to define DB2-ALIAS members are optional. However, for the successful generation of SQL statements, you must define specific clauses, as follows:

- for CREATE ALIAS statements define the CREATOR-OWNER clause
- for DROP ALIAS statements define the CREATOR-OWNER clause
- for COMMENT ON statements define the DB2-COMMENT clause
- for LABEL ON statements define the DB2-LABEL clause.

Note: the DB2-ALIAS member type is not the same as the common clause ALIAS, which enables you to define alternative names for a repository member.

The more commonly used relationships to and from DB2-ALIAS members are:



AS

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined in another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

CREATOR-OWNER

To define the owner of an object, enter:

```
CREATOR-OWNER user
```

user is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the object. On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member. The CREATOR-OWNER clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in an export to DB2 panel.

Note: the DB2-USER or SQL-USER member named in the CREATOR-OWNER clause is used to generate user-qualified names for tables, views and aliases. If the DB2-USER has a LOCATION clause defined, and the DB2 profile is set to three-part name generation, location and user-qualified names are generated for tables and views. The location-qualifier can be overridden by a LOCATION clause defined in an export to DB2 panel.

FOR
To specify the table or view to which the DB2 alias applies, enter:

```
FOR object
```

object is a member of one of the following types:

- DB2-TABLE
- DB2-VIEW
- SQL-TABLE
- SQL-VIEW.

On encoding, the member specified in the FOR clause is checked to ensure that it is one of the above.

For the successful generation of SQL CREATE ALIAS and DROP ALIAS statements the member named in the FOR clause:

- must be encoded
- must have a CREATOR-OWNER clause defined, naming a DB2-USER member
- if your DB2 profile is set to three-part name generation, the DB2-USER member must have a LOCATION clause defined.

DB2-COMMENT
To define a comment for an alias, table, view, or column, enter a string of no more than 254 characters.
When generated, lines of comment are concatenated to form a single string. To preserve spaces between words, insert a space after the last character of each continuing line.
For example, the DB2-TABLE named MANAGER-NUMBER has an owner of PERSONNEL and the following comment defined:

> This table contains the Manager number of every manager in each department.

The following SQL statement can be generated:

```
COMMENT ON TABLE PERSONNEL.MANAGER_NUMBER IS 'This table cont
ains the Manager number of every manager in each department'.
```

In this example the word "contains" has been split due to the margins set in the DB2 profile.

This clause must be present for the successful generation of SQL COMMENT ON statements.

Note:

- for columns, the comment definition must follow the CONTAINS clause defining that column or group of columns

- for tables and views, the comment definition should precede the COLUMNS clause that defines the columns of the table.

DB2-LABEL
To define a label for an alias, table, view or column, enter a string of no more than 30 characters.
This clause must be present for the successful generation of SQL LABEL ON statements.

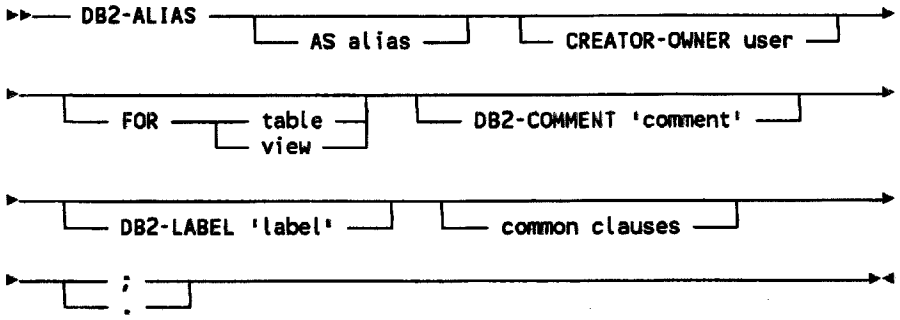
DB2-ALIAS

Note:

- for columns, the label definition must follow the CONTAINS clause defining that column or group of columns

- for tables and views, the label definition should precede the COLUMNS clause that defines the columns of the table.

Syntax



where

alias is the name of a DB2-ALIAS member

user is the name of a DB2-USER or SQL-USER member

table is the name of a DB2-TABLE or SQL-TABLE member

view is the name of a DB2-VIEW or SQL-VIEW member

comment is a string of no more than 254 characters, within delimiters

label is a string of no more than 30 characters, within delimiters

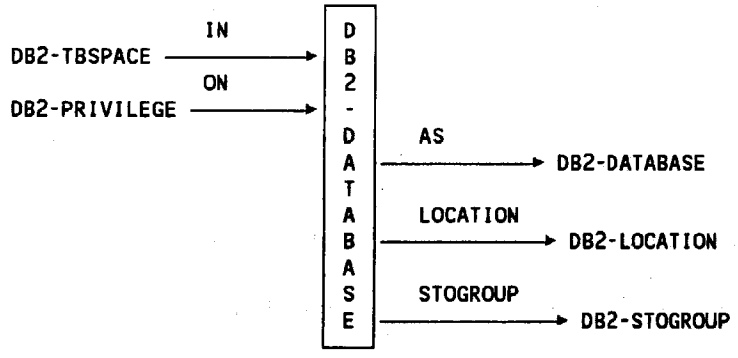
common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DB2-DATABASE

All the clauses available to define DB2-DATABASE members are optional. You can generate SQL CREATE DATABASE and DROP DATABASE statements from DB2-DATABASE members.

The more commonly used relationships to and from DB2-DATABASE are:



AS

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

LOCATION
To define the location to which a database or user belongs, enter:

```
LOCATION location-name
```

location-name is the name of a DB2-LOCATION member that represents a local or remote location in a distributed network.

In DB2-DATABASE member definitions, the LOCATION clause is available to document your DB2 environment but is not used to generate location-qualified names.

STOGROUP
To define the storage group to which the database belongs, enter:

```
STOGROUP stogroup-name
```

stogroup-name is the name of a DB2-STOGROUP member.
On encoding, the member specified in the STOGROUP clause is checked to ensure that it is a DB2-STOGROUP member.
The storage group defined in the DB2-DATABASE definition is the default used by generated table spaces and indexes:

- that belong to the database
- that do not have a storage group specified in their own member definition.

If there is no storage group defined in the DB2-DATABASE member, none is generated and the DB2 default, SYSDEFLT, applies. However you are recommended to explicitly define the storage group, even if it is SYSDEFLT, so that the repository accurately reflects your DB2 environment.

BUFFERPOOL
To define the buffer pool that databases use, enter:

```
BUFFERPOOL bufferpool-name
```

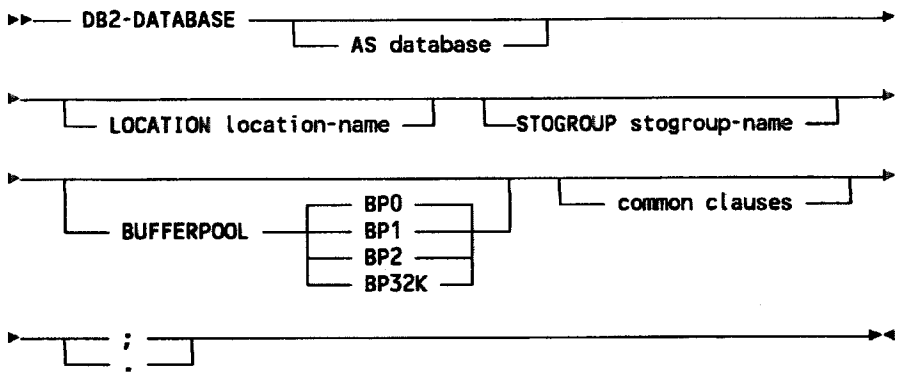
bufferpool-name is one of the following buffer pools:

```
BP0             BP1
BP2             BP32K
```

The buffer pool defined in the DB2-DATABASE definition is the default used by those table spaces and indexes that belong to the database, and do not have a buffer pool specified in their own member definition.

If you do not define a buffer pool in a DB2-DATABASE, none is generated and the DB2 default applies.

Syntax



where

database is the name of a DB2-DATABASE member

location-name is the name of a DB2-LOCATION member

stogroup-name is the name of a DB2-STOGROUP member

common clauses are any of the clauses common to all member types. Refer to Appendix 2 for details of the common clauses.

DB2-INDEX

All the clauses available to define DB2-INDEX members are optional. However for the successful generation of SQL statements you must define the following:

- for ALTER INDEX statements define the CREATOR-OWNER clause
- for CREATE INDEX statements define the CREATOR-OWNER and ON clauses, and if the PARTITION clause is defined, the CLUSTER keyword
- for DROP INDEX statements define the CREATOR-OWNER clause.

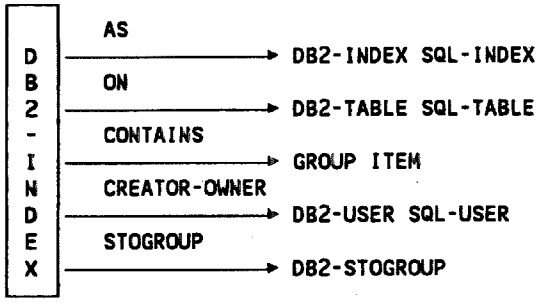
Each column is usually represented by an ITEM member, but can be represented by a GROUP member. GROUP members are useful because they can contain several ITEMs.

To specify the ITEM and GROUP members that represent the indexed columns, use the CONTAINS clause. You can define columns:

- individually, so that one ITEM or GROUP member defines one column
- in sets, so that the same ITEM or GROUP member defines several columns, with identical attributes
- in cascades from a GROUP member, so that every ITEM nested in a GROUP member defines one column.

DB2-INDEX repository definitions can be generated automatically if you use the Workbench Design Area (WBDA) facilities for DB2 database design.

The more commonly used relationships to and from DB2-INDEX are:



AS

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

CREATOR-OWNER
To define the owner of an object, enter:

```
CREATOR-OWNER user
```

user is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the object.

On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member. The CREATOR-OWNER clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in an export to DB2 panel.

Note: the DB2-USER or SQL-USER member named in the CREATOR-OWNER clause is used to generate user-qualified names for tables, views and aliases. If the DB2-USER has a LOCATION clause defined, and the DB2 profile is set to three-part name generation, location and user-qualified names are generated for tables and views. The location-qualifier can be overridden by a LOCATION clause defined in an export to DB2 panel.

UNIQUE
To define that indexed columns in the table being indexed do not have duplicate entries, enter:

```
UNIQUE
```

In DB2, when a single column in a table is being indexed, then each value can appear once only in the indexed column. When more than one column in a table is being indexed, then any given set of values can appear once only in the indexed columns.

ON
To define the table to which the index refers, enter:

```
ON table
```

table is the name of a DB2-TABLE or SQL-TABLE member.
On encoding, the member specified in the ON clause is checked to ensure that it is a DB2-TABLE or SQL-TABLE member.
The ON clause must be present for the successful generation of SQL CREATE INDEX statements.

DEFAULTED-AS
To specify that the DEFAULTED-AS form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns, enter:

```
DEFAULTED-AS
```

and delete:

```
ENTERED-AS
HELD-AS
REPORTED-AS
```

The form keyword that you define applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.
If you do not specify a form keyword then the DEFAULTED-AS form description is used.

HELD-AS
To specify that the HELD-AS form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns, enter:

```
HELD-AS
```

and delete:

```
ENTERED-AS
REPORTED-AS
DEFAULTED-AS
```

The form keyword that you define applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.
If you do not specify a form keyword then the DEFAULTED-AS form description is used.

ENTERED-AS
To specify that the ENTERED-AS form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns, enter:

```
ENTERED-AS
```

and delete:

```
HELD-AS
REPORTED-AS
DEFAULTED-AS
```

The form keyword that you define applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.
For example, a DB2-TABLE, DB2-INDEX or DB2-VIEW member containing the following lines:

```
ENTERED-AS
CONTAINS ITEM1, ITEM2
```

refers to the two ITEM members:

ITEM1

```
ITEM
HELD-AS BINARY 10
ENTERED-AS CHAR 5
REPORTED-AS FLOAT 9
```

ITEM2

```
ITEM
ENTERED-AS DECIMAL 4.2
DEFAULTED-AS FLOAT 7
```

The ENTERED-AS form keyword in the DB2-INDEX or DB2-VIEW definition specifies that the ENTERED-AS form description from both ITEMs is used to define the data type of columns.
Therefore the column generated from ITEM1 has a data type of CHAR and the column generated from ITEM2 has a data type of DECIMAL.

If you do not specify a form keyword then the DEFAULTED-AS form description is used. Where the ITEM or GROUP has no DEFAULTED-AS form description defined, then the MANAGER Products defaults apply.

REPORTED-AS
To specify that the REPORTED-AS form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns, enter:

```
REPORTED-AS
```

and delete:

```
HELD-AS
ENTERED-AS
DEFAULTED-AS
```

The form keyword that you define applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.
If you do not specify a form keyword then the DEFAULTED-AS form description is used.

CONTAINS
To specify the ITEM or GROUP members that define columns, enter:

```
CONTAINS member-list
```

member-list is the name of one or more ITEM or GROUP members, separated by commas, each representing a single column.

On encoding, the members specified in the CONTAINS clause are checked to ensure that they are either ITEMs or GROUPs. Duplicate column names are not permitted by DB2, therefore column names are checked on generation to ensure that no duplicates are present.

Each ITEM can define up to 15 form descriptions. To define which of the form descriptions you want to use, enter:

```
CONTAINS item version
```

item is the name of an ITEM member.

version is an integer in the range 1 to 15, and defines the form description version that you want to use.

For example:

```
HELD-AS CONTAINS STOCK-LIST 3
```

defines that the third HELD-AS form description in the ITEM member STOCK-LIST is used as the column data type.

To define a set of columns with identical attributes, using the same ITEM or GROUP member, enter:

```
CONTAINS (integer) member
```

integer is the number of columns to be derived from the member, within brackets.

member is the name of an ITEM or GROUP member.

To resolve duplicated names on generation of an SQL statement, column names are automatically suffixed by an underscore and a number, the first by_1, the second by_2 and so on.

For example:

```
CONTAINS (4) STOCK-LIST
```

generates the four columns STOCK_LIST_1, STOCK_LIST_2, STOCK_LIST_3 and STOCK_LIST_4. The generated attributes, such as data type, are the same for each of the four columns.
When a column is derived from a GROUP containing several ITEMs, the data type of the column is taken as CHAR. The maximum number of characters allowed in the column is calculated from the combined field lengths of the form descriptions defined in each ITEM. However, where an export to DB2 panel applied to the DB2-TABLE specifies the EXPAND keyword, then each ITEM within a GROUP generates a separate column.

VCAT
To define a control or master level password used to access the VSAM catalog, enter:

```
PASSWORD password
```

password is a control or master level VSAM catalog password, of no more than eight characters.

When you generate SQL statements, the VCAT password is taken directly from the password you specify in this clause.

STOGROUP
To define the physical space occupied by an index or partition you can either:

- define the VSAM catalog it is to use

or:

- define the storage group it belongs to.

To define the VSAM catalog the index or partition is to use, enter:

```
VCAT catalog
```

catalog is the name of a VSAM catalog, of no more than 8 characters.

To define the storage group to which the index or partition belongs, enter:

```
STOGROUP stogroup-name
```

stogroup-name is the name of a DB2-STOGROUP member.

On encoding, the member specified in the STOGROUP clause is checked to ensure that it is a DB2-STOGROUP member.

The storage group defined in the DB2-DATABASE definition is the default used by indexes:

- that belong to the database
- that do not have a storage group specified in their own member definition.

FREEPAGE
You can accommodate future expansion of an index, table space or partition by defining the frequency with which pages are left free.

To define the relative frequency with which free pages are allocated, enter:

```
FREEPAGE fn
```

fn is an integer in the range 0 to 255.

For example, FREEPAGE 4 means that 1 free page is left after every 4 pages.

If you do not define a FREEPAGE clause the DB2 defaults apply.

PCTFREE
You can accommodate future expansion of an index, table space or partition by defining the percentage of each page that is left free.

To define the percentage space kept free on a page, when an index, table space or partition is loaded or reorganized, enter:

```
PCTFREE pn
```

pn is an integer in the range 0 to 99 .

If you do not define a PCTFREE clause the DB2 defaults apply.

CLUSTER
To define a clustered index, enter:

```
CLUSTER
```

Partitioned indexes must also be clustered.
If you do not include the CLUSTER keyword in a DB2-INDEX that contains a PARTITION clause, it is automatically generated in SQL CREATE INDEX statements. A warning message is also generated, to remind you to include the CLUSTER keyword in the member definition.

PARTITION
To define that an index or table space is to be partitioned, enter:

```
PARTITION
```

You can optionally define a number, details of storage, free space allocation and key values, for each partition.

To successfully generate SQL CREATE INDEX statements from DB2-INDEX members containing a PARTITION clause, you must define a NUMBER and KEY clause for each PARTITION.

To give the partition a number, enter:

```
NUMBER n
```

n is an integer in the range 1 to 64.

To define that the partitions of an index have a key value, enter:

```
KEY 'key-val'
```

key-val is the highest value, within delimiters, that a column in the partition can have.

If you are indexing more than one column, each key value must be enclosed in quotes and separated by a comma. The first key value corresponds to the first index column, the second value to the second index column and so on.
If the key value is a character field it must be enclosed in single quotes within double quotes. For example:

```
KEY " 'key-val' "
```

To define the storage space to which the partition belongs, use the VCAT, or STOGROUP, PRIQTY, SECQTY and ERASE clauses. To define how much space is left free in the partition, use the FREEPAGE and PCTFREE clauses.

If you do not define storage or free space for partitions individually, they automatically take space from that specified for the index or table space as a whole.

SUBPAGES
Each physical page of an index may be divided into 1, 2, 4, 8 or 16 subpages. Each subpage is a unit of locking.

To define subpages and therefore the locking unit, enter:

```
SUBPAGES division
```

division is one of the following integers:

1
2
4
8
16

If you do not define a locking parameter, none is generated, and the DB2 default applies.

BUFFERPOOL
To define the buffer pool that indexes use, enter:

```
BUFFERPOOL bufferpool-name
```

bufferpool-name is one of the following buffer pools:

```
BP0          BP1
BP2          BP32K
```

The buffer pool defined in the DB2-DATABASE definition is the default used by those indexes that belong to the database, and do not have a buffer pool specified in their own member definition.

CLOSE
To define that the data set, on which an index or table space resides, is to be closed when not in use, enter:

```
CLOSE YES
```

To define that it is to remain open, enter:

```
CLOSE NO
```

If you do not define a CLOSE clause the DB2 default applies.

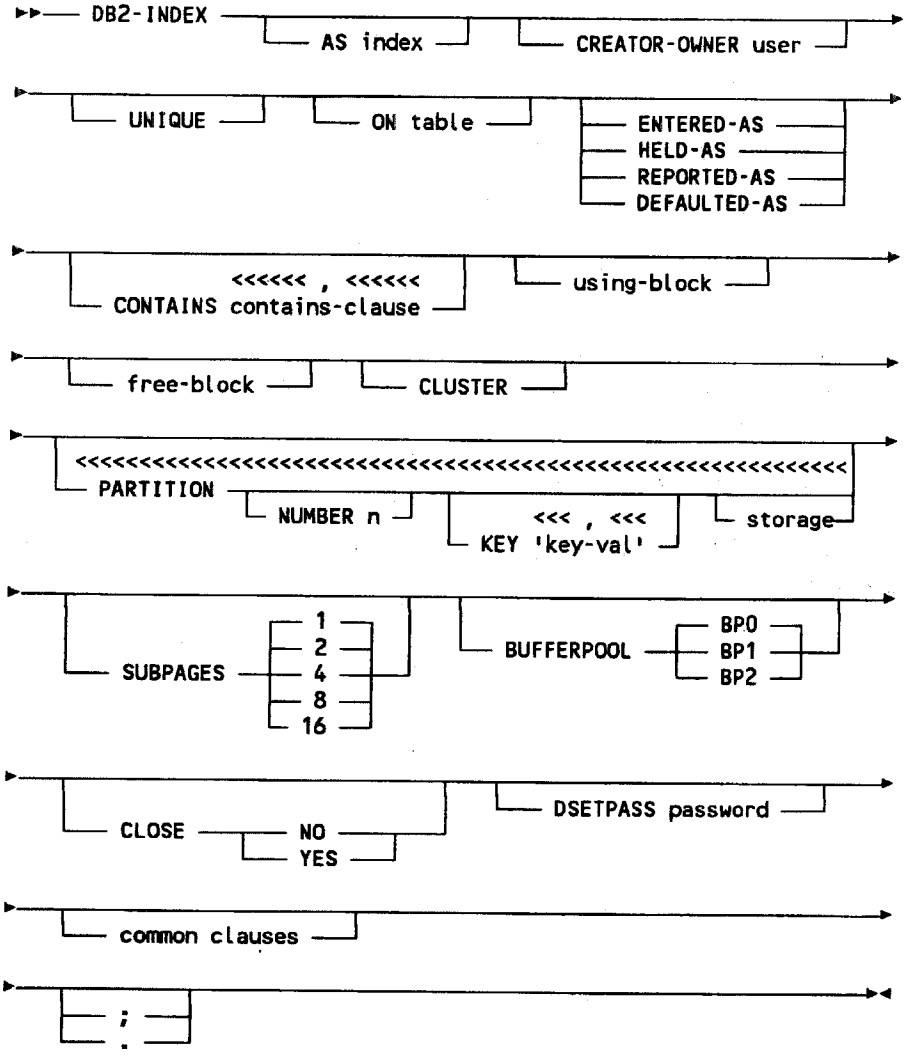
DSETPASS
To define a password for the VSAM data set, on which an index or table space resides, enter:

```
DSETASS password
```

password is a VSAM data set password of no more than 8 characters.

DB2-INDEX

Syntax



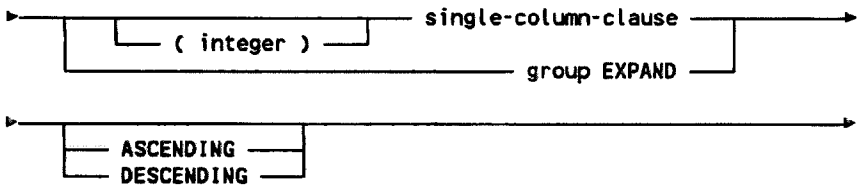
where

index is the name of a DB2-INDEX or SQL-INDEX member

user is the name of a DB2-USER or SQL-USER member

table is the name of a DB2-TABLE or SQL-TABLE member

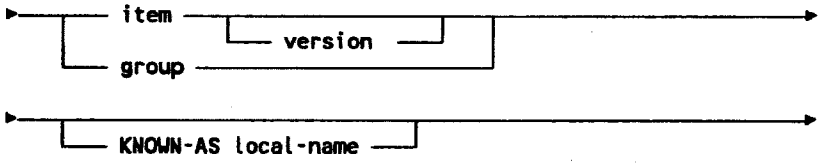
contains-clause is:



where

integer is the number of columns in a set

single-column-clause is:



where

item is the name of an ITEM member

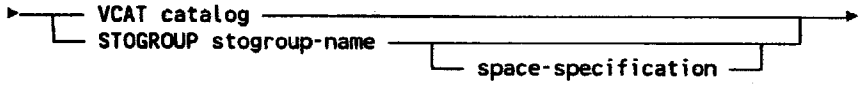
version is an integer in the range 1 to 15

group is the name of a GROUP member

local-name is the name of the column, of no more than 18 characters.

group is as defined above.

using-block is:

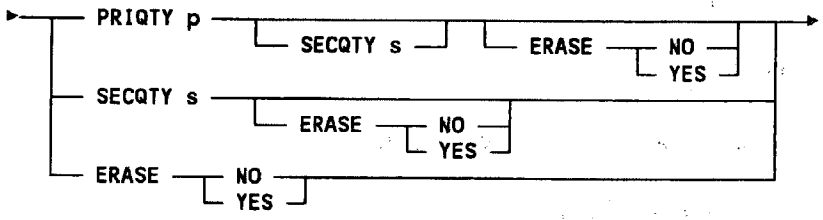


where

catalog is a VSAM catalog name, of no more than 8 characters

stogroup-name is the name of a DB2-STOGROUP member

space-specification is:

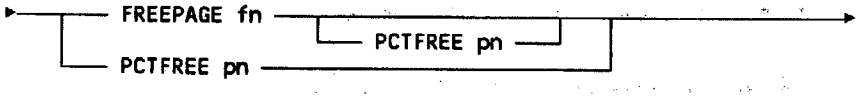


where

p is an integer in the range 3 to 4194304

s is an integer in the range 0 to 131068.

free-block is:



where

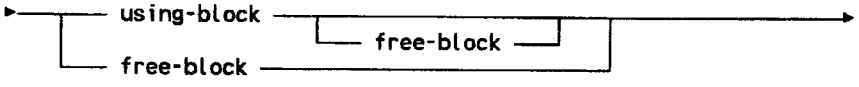
fn is an integer in the range 0 to 255

pn is an integer in the range 0 to 99.

n is an integer in the range 1 to 64

key-val is the highest value which the column may contain in the partition

storage is:



where

using-block is as defined above

free-block is as defined above.

password is a VSAM data set password, of no more than 8 characters

common clauses are any of the clauses common to all member types. Refer to Appendix 2 for details of the common clauses.

DB2-LOCATION

Use the DB2-LOCATION member type to document a specific location in a distributed network, which represents a DB2 subsystem.

DB2-LOCATION members are used to generate three-part qualified object names when your DB2 profile is set for three-part name generation. Two-part names are generated by default.
To generate a location qualifier for table and view names you can:

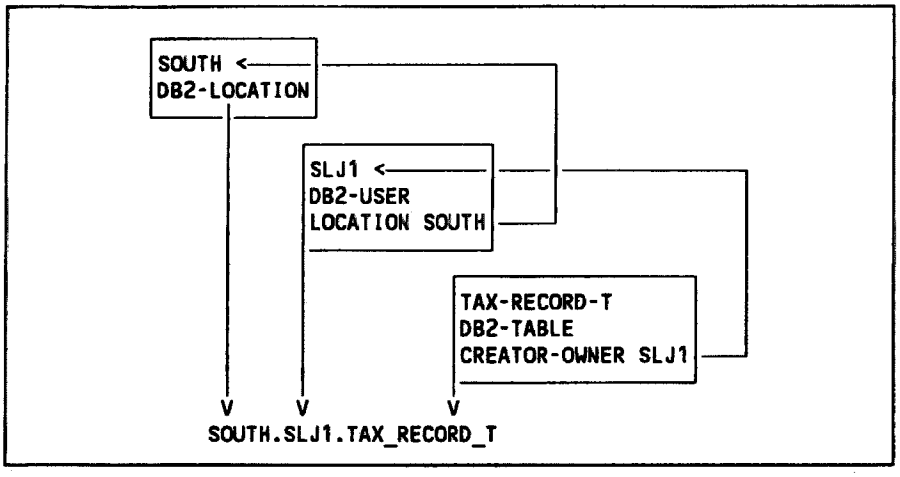
- use export to DB2 panels with the LOCATION keyword (for example DB2 CREATE)
- use relationships between repository members.

Note: a location explicitly named in an export to DB2 panel overrides one derived from member relationships.
You can generate a location qualifier from the following relationships between members:

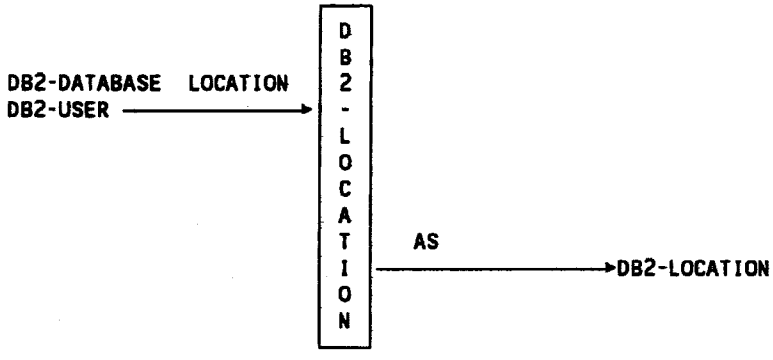
- a DB2-LOCATION member must be named in the LOCATION clause of a DB2-USER member

- the DB2-USER member must be named as the CREATOR-OWNER of the DB2-TABLE or DB2-VIEW being generated.

For example, to generate the three-part table name SOUTH.SLJ1.TAX_RECORD_T, the following relationships must exist between members in the repository:



The more commonly used relationships to and from DB2-LOCATION are:



AS

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

DB2-LOCATION

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

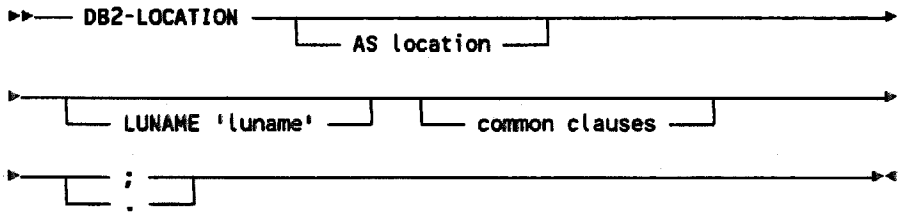
During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

LUNAME
To document the VTAM name of a DB2 location, enter:

```
LUNAME luname
```

luname is a string of no more than eight characters and represents the VTAM logical unit name of the DB2 location. The LUNAME clause is not used to generate SQL statements, but is available to document your DB2 database.

Syntax



where

location is the name of a DB2-LOCATION member

luname is the VTAM name for a location in a network. It is a maximum of 8 characters and must be in delimiters

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

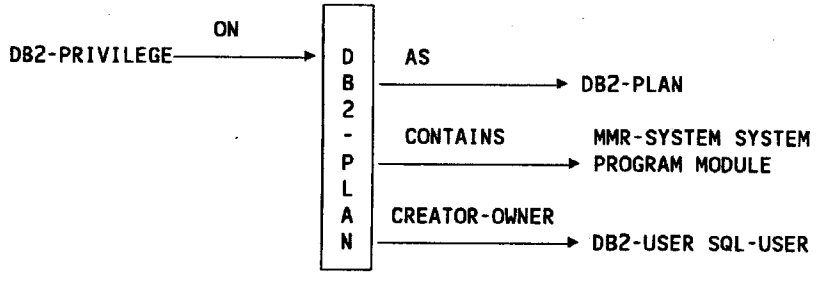
DB2-PLAN

All the clauses available to define DB2-PLAN members are optional. However the CREATOR-OWNER and CONTAINS clauses must be defined for the successful generation of BIND and REBIND subcommands.

Usually MODULE members are used to represent modules of code containing embedded SQL statements, but PROGRAM or SYSTEM members can also be used. The source code modules are passed through the DB2 Precompiler to produce precompiled code modules and Database Request Modules (DBRMs). The BIND or REBIND subcommand is used to bind together the DBRMs and so produce the application plan in the DB2 database. The precompiled code is compiled and link-edited to form the application program that uses the plan to access DB2.

The BIND and REBIND subcommands can be generated from DB2-PLAN definitions using the DB2 BIND and DB2 REBIND commands respectively.

The more commonly used relationships to and from DB2-PLAN members are:



AS
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

CREATOR-OWNER
To define the owner of an object, enter:

```
CREATOR-OWNER user
```

user is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the object.
On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member. The CREATOR-OWNER clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in an export to DB2 panel.

Note: the DB2-USER or SQL-USER member named in the CREATOR-OWNER clause is used to generate user-qualified names for tables, views and aliases. If the DB2-USER has a LOCATION clause defined, and the DB2 profile is set to three-part name generation, location and user-qualified names are generated for tables and views. The location-qualifier can be overridden by a LOCATION clause defined in an export to DB2 panel.

CONTAINS
To specify the members that represent the database request modules bound (DBRM) into the plan, enter:

```
CONTAINS member-list
```

member-list is the name of one or more MODULE, PROGRAM, MMR-SYSTEM or SYSTEM members, separated by commas. On encoding, the members specified in the CONTAINS clause are checked to ensure that they are MODULE, PROGRAM, MMR-SYSTEM or SYSTEM members.

PREPARE
To define parameters within which the BIND or REBIND subcommands produce the application plan, enter either:

```
DEFER    or    NODEFER
```

You can only define one option. The clauses correspond to the keywords in the BIND and REBIND subcommands defined in the IBM documentation and their meanings are the same. Additional parameters can be defined using the DB2 BIND or DB2 REBIND commands.

VALIDATE
To define parameters within which the BIND or REBIND subcommands produce the application plan, enter either:

```
RUN    or    BIND
```

You can only define one option for each parameter. The clauses correspond to the keywords in the BIND and REBIND subcommands defined in the IBM documentation and their meanings are the same. Additional parameters can be defined using the DB2 BIND or DB2 REBIND commands.

ISOLATION
To define parameters within which the BIND or REBIND subcommands produce the application plan, enter either:

```
RR   or   CS
```

You can only define one option for the parameter. The clauses correspond to the keywords in the BIND and REBIND subcommands defined in the IBM documentation and their meanings are the same. Additional parameters can be defined using the DB2 BIND or DB2 REBIND commands.

Note: for the successful generation of BIND and REBIND subcommands you must define RELEASE DEALLOCATE with ACQUIRE ALLOCATE.

ACQUIRE
To define the ACQUIRE parameters within which the BIND or REBIND subcommands produce the application plan, enter either:

```
USE
```

or

```
ALLOCATE
```

You can only define one option. The clause corresponds to the keywords in the BIND and REBIND subcommands defined in the IBM documentation and their meanings are the same. Additional parameters can be defined using the DB2 BIND or DB2 REBIND commands.

Note: for the successful generation of BIND and REBIND subcommands you must define RELEASE DEALLOCATE with ACQUIRE ALLOCATE.

RELEASE
To define parameters within which the BIND or REBIND subcommands produce the application plan, enter either:

```
COMMIT      or      DEALLOCATE
```

You can only define one option. The clause corresponds to the keywords in the BIND and REBIND subcommands defined in the IBM documentation and their meanings are the same. Additional parameters can be defined using the DB2 BIND or DB2 REBIND commands.

Note: for the successful generation of BIND and REBIND subcommands you must define RELEASE DEALLOCATE with ACQUIRE ALLOCATE.

EXPLAIN
To define parameters within which the BIND or REBIND subcommands produce the application plan, enter either:

```
YES
```

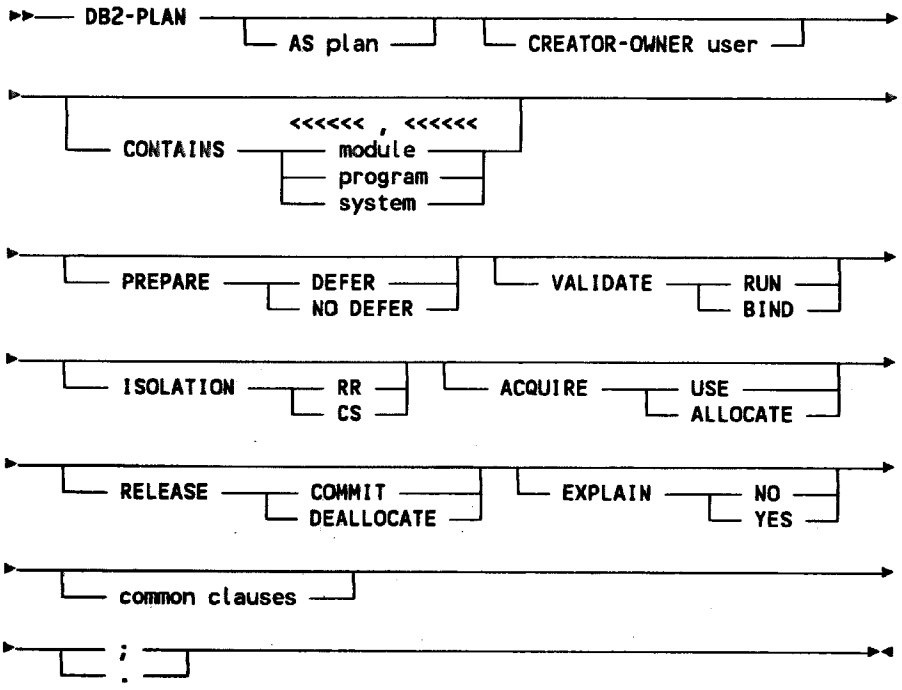
or

```
NO
```

You can only define one option for the parameter. The clauses correspond to the keywords in the BIND and REBIND subcommands defined in the IBM documentation and their meanings are the same. Additional parameters can be defined using the DB2 BIND or DB2 REBIND commands.

Note: for the successful generation of BIND and REBIND subcommands you must define RELEASE DEALLOCATE with ACQUIRE ALLOCATE.

Syntax



where

member is the name of a DB2-PLAN member

user is the name of a DB2-USER or SQL member

module is the name of a MODULE member

program is the name of a PROGRAM member

system is the name of a SYSTEM or MMR-SYSTEM member

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DB2-PRIVILEGE

All the clauses available to define DB2-PRIVILEGE members are optional.

You can define privileges on:

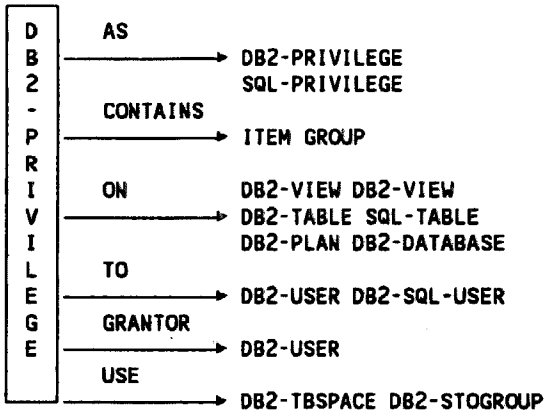
- databases
- tables
- plans
- buffer pools
- storage groups
- table spaces
- the DB2 system.

You can generate SQL GRANT or REVOKE statements from DB2-PRIVILEGE members using the DB2 GRANT and DB2 REVOKE commands. All privileges, except SYSTEM and USE privileges, give access to particular DB2 objects. For example, a DATABASE privilege gives a user access to a specific, named database, defined in the ON clause.

You can record the grantors and recipients of privileges, and optionally define that the recipient may pass on the privilege to another user.

Only remove a DB2-PRIVILEGE member from the repository if you have generated an SQL REVOKE statement and do not need the member any more. If you need to grant the privilege again, retain the member, so that the repository reflects your long-term security arrangements.

The more commonly used relationships to and from DB2-PRIVILEGE are:



AS

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

DB2-PRIVILEGE

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

GRANTOR

To define the user who is granting the privilege, enter:

```
GRANTOR user
```

user is the name of a DB2-USER or SQL-USER member and represents the authorization ID of the user who is granting the privilege. The grantor is usually a database administrator for a project.

On encoding the GRANTOR clause is checked to ensure that the member named is a DB2-USER or SQL-USER member. The clause is not used in the generation of SQL statements and is for documentation purposes only.

DATABASE

To define privileges on databases, enter:

```
DATABASE
```

followed by one, several or all of the following optional keywords that correspond to privileges allowed in DB2:

```
DBADM       DBCTRL
DBMAINT     CREATETAB
CREATETS    DISPLAYDB
DROP        IMAGCOPY
LOAD        RECOVERDB
REORG       REPAIR
STARTDB     STOPDB
STATS
```

The privileges are identified by the same keywords as those defined in the IBM documentation, and have the same meaning. On encoding, the DATABASE clause is checked to ensure that the privileges specified are valid.

For the successful generation of SQL statements, the DB2-DATABASE to which the privilege applies must be defined using an ON clause.

PLAN
To define privileges on plans, enter:

```
PLAN
```

followed by one or both of the following privileges:

```
BIND
EXECUTE
```

The privileges are identified by the same keywords as those defined in the IBM documentation, and have the same meaning. On encoding, the PLAN clause is checked to ensure that the privileges specified are valid.

For the successful generation of SQL statements, the DB2-PLAN to which the privilege applies must be defined using an ON clause.

TABLE
To grant privileges on a table or view, enter:

```
TABLE
```

followed by one, several or all of the following privileges:

```
ALTER           DELETE
INDEX           INSERT
SELECT          UPDATE
```

DB2-PRIVILEGE

The privileges are identified by the same keywords as those defined in the IBM documentation, and have the same meaning. To optionally define the columns that the UPDATE privilege applies to in the table or view, enter:

```
UPDATE column-definition
```

column-definition is either:

```
(integer) member KNOWN-AS local-name
```

or:

```
group EXPAND
```

integer is the number of columns to be generated from the member.

member is the name of an ITEM or GROUP member.

local-name is the name of the column.

group is the name of a GROUP member.

On encoding, the TABLE clause is checked to ensure that the privileges specified are valid.

For the successful generation of SQL statements, the DB2-TABLE or DB2-VIEW to which the privilege applies must be defined using an ON clause.

ALL
To grant all the privileges on a table or view, enter:

```
ALL
```

For the successful generation of SQL statements, the DB2-TABLE or DB2-VIEW to which all the table privileges apply must be defined using an ON clause.

SYSTEM
To define privileges to monitor and maintain the system, enter:

```
SYSTEM
```

followed by one, several or all of the following privileges:

```
SYSADM          SYSOPR
BINDADD         BSDS
CREATEALIAS     CREATEDBA
CREATEDBC       CREATESG
DISPLAY         RECOVER
STOPALL         STOSPACE
TRACE           MONITOR1
MONITOR2
```

The privileges are identified by the same keywords as those defined in the IBM documentation, and have the same meaning. On encoding, the SYSTEM clause is checked to ensure that the privileges specified are valid.

USE
To specify the privilege to use buffer pools, enter:

```
USE BUFFERPOOL bufferpool-name
```

bufferpool-name is one or more of the following buffer pools, separated by commas:

```
BP0             BP1
BP2             BP32K
```

To specify the privilege to use named storage groups, enter:

```
USE STOGROUP stogroup-name
```

stogroup-name is the name of one or more DB2-STOGROUP members, separated by commas.

To specify the privilege to use named table spaces, enter:

```
USE TABLESPACE tbspace-name
```

tbspace-name is the name of one or more DB2-TBSPACE members, separated by commas.

Note: you can specify only one USE clause.
On encoding, the USE clause is checked to ensure that the privilege specified is valid.

TO
To define the user to whom the privilege applies, enter:

```
TO user
```

user is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the user.

To define that the privilege applies to all users, enter:

```
TO PUBLIC
```

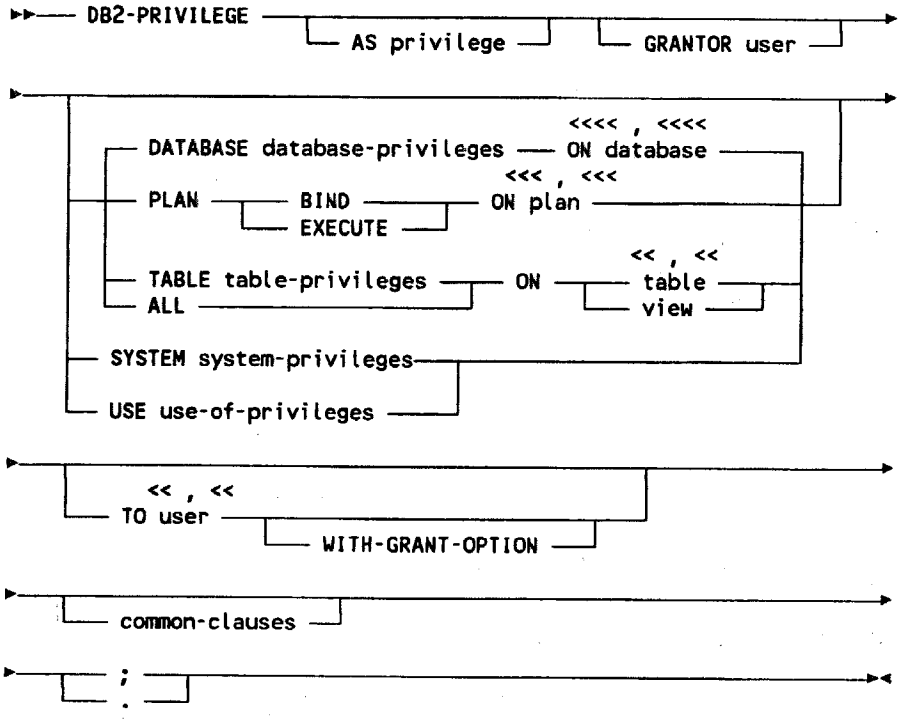
If you want to interrogate the repository to find out which privileges are granted to all users, define a DB2-USER member named PUBLIC.

To optionally specify that the recipient(s) of the privilege may transfer it to another user, enter:

```
WITH-GRANT-OPTION
```

The TO clause must be present for the successful generation of an SQL statement.

Syntax



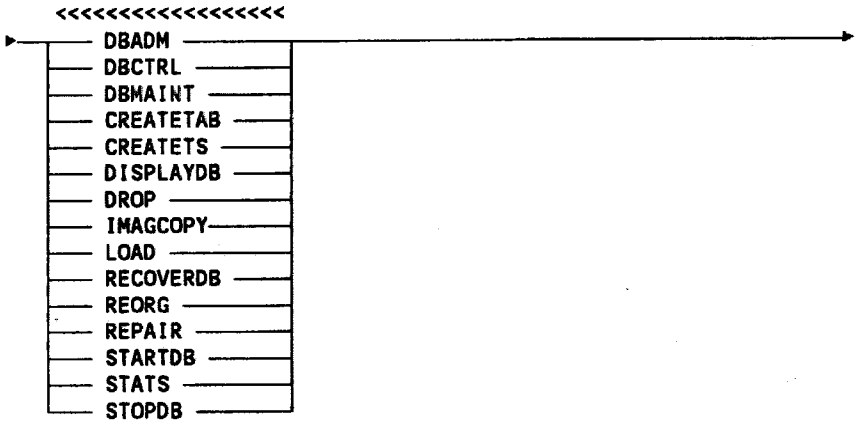
where

privilege is the name of a DB2-PRIVILEGE or SQL-PRIVILEGE member

user is the name of a DB2-USER or SQL-USER member

DB2-PRIVILEGE

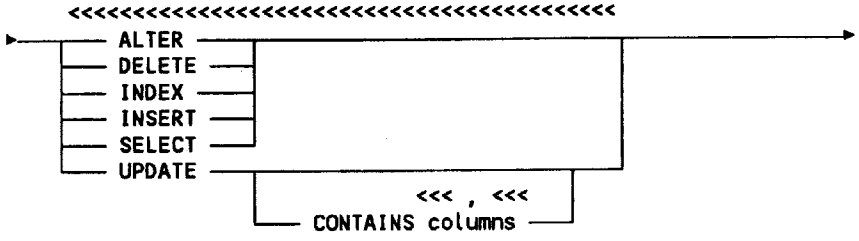
database-privileges are:



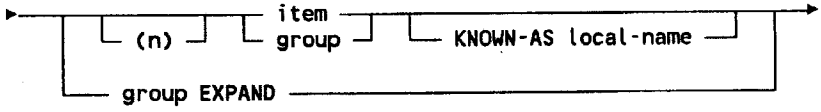
database is the name of a DB2-DATABASE member

plan is the name of a DB2-PLAN member

table-privileges are:



where **columns** are:



where

n is the number of columns in a column set

item is the name of an ITEM member

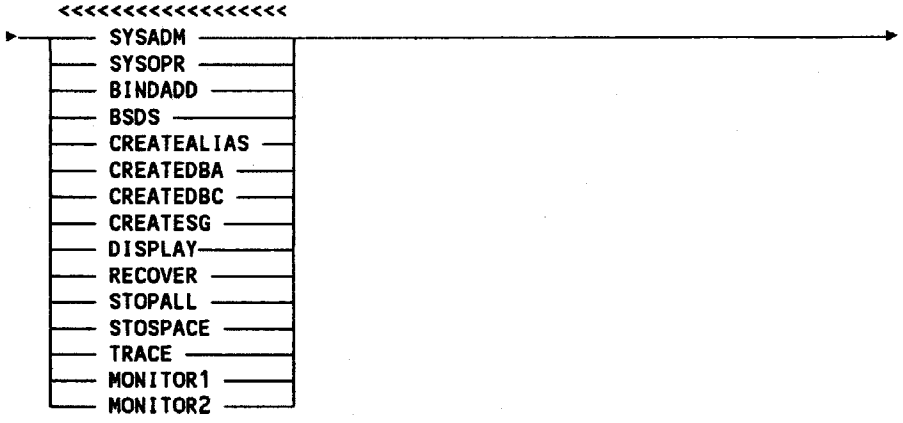
group is the name of a GROUP member

local-name is a name, of no more than 18 characters.

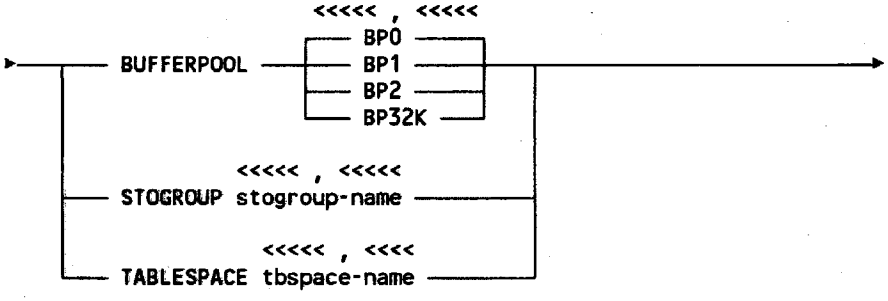
table is the name of a DB2-TABLE or SQL-TABLE member

view is the name of a DB2-VIEW or SQL-VIEW member

system-privileges are:



use-of-privileges are:



where

stogroup-name is the name of a DB2-STOGROUP member

tbspace-name is the name of a DB2-TBSPACE member.

DB2-PRIVILEGE

user is as defined above

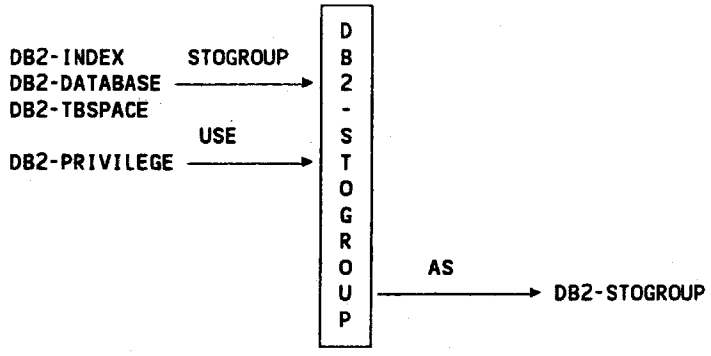
common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DB2-STOGROUP

All the clauses available to define DB2-STOGROUP members are optional. However, the VOLUMES and VCAT clauses must be present for the successful generation of SQL CREATE STOGROUP statements.

The more commonly used relationships to and from DB2-STORAGE-GROUP are :



AS
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

DB2-STOGROUP

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

VOLUMES
To define the DASD volumes on which the storage group resides, enter:

```
VOLUMES vol-id-list
```

vol-id-list is one or more storage volume names, each of no more than six characters, and separated by commas. You can define a maximum of 133 storage volume names.
At least one volume must be defined for the successful generation of an SQL CREATE STOGROUP statement.
When you generate SQL statements, DB2 names for volumes are taken directly from the names you specify in this clause.

VCAT
To define a control or master level password used to access the VSAM catalog, enter:

```
PASSWORD password
```

password is a control or master level VSAM catalog password, of no more than eight characters.
When you generate SQL statements, the VCAT password is taken directly from the password you specify in this clause.

PASSWORD
To define a control or master level password used to access the VSAM catalog, enter:

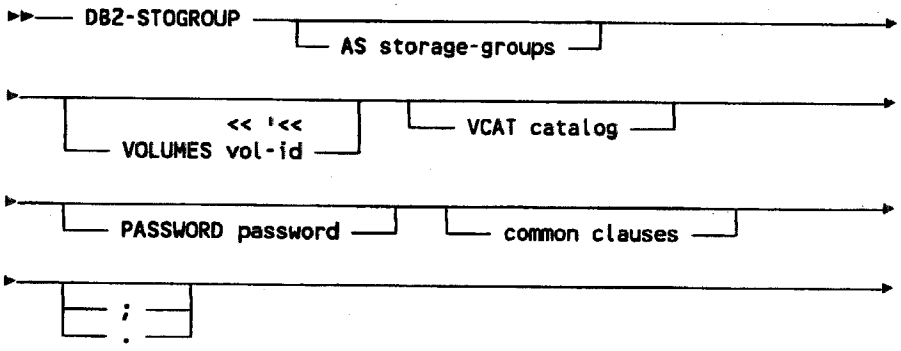
```
PASSWORD password
```

password is a control or master level VSAM catalog password, of no more than eight characters.

When you generate SQL statements, the VCAT password is taken directly from the password you specify in this clause.

DB2-STOGROUP

Syntax



where

storage-groups is the name of a DB2-STOGROUP member

vol-id is a storage volume name, of no more than 6 characters

catalog is a VSAM catalog name, of no more than 8 characters

password is a VSAM catalog password, of no more than 8 characters

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DB2-TABLE

Tables are of major interest to the end-user, since they contain the user's own data and are not a product of the database administrator or other technical specialist. Therefore the DB2-TABLE is one of the most used DB2 member types. All the clauses available to define DB2-TABLE members are optional. However, for the successful generation of SQL statements you must define specific clauses, as follows:

- for ALTER TABLE statements define the CREATOR-OWNER clause
- for CREATE TABLE statements define the CREATOR-OWNER, COLUMNS and IN clauses
- for COMMENT ON statements define the CREATOR-OWNER and DB2-COMMENT clauses
- for DECLARE TABLE statements define the CREATOR-OWNER and COLUMNS clauses
- for DROP TABLE statements define the CREATOR-OWNER clause
- for LABEL ON statements define the CREATOR-OWNER and DB2-LABEL clauses.

To specify the ITEM and GROUP members that represent the columns of the table, use the CONTAINS clause. It establishes relationships between a DB2-TABLE and ITEM and GROUP members, which define the table's columns. These GROUPs and ITEMs may also form part of other file and database segment definitions. For example, installations with IMS may already have GROUP and ITEM definitions in the repository, which can now be shared with the DB2 environment.
You can define columns:

- individually, so that one ITEM or GROUP member defines one column
- in sets, so that the same ITEM or GROUP member defines several columns, with identical attributes
- in cascades from a GROUP member, so that every ITEM nested in a GROUP member defines one column.

DB2-TABLE

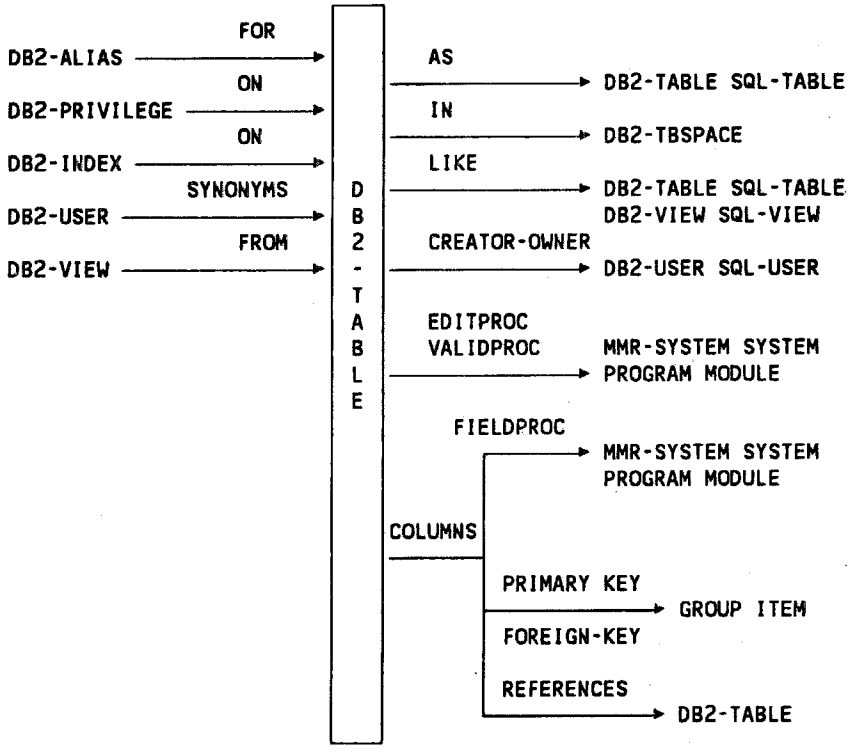
Sub-clauses within the COLUMNS clause enable you to define:

- the names of columns
- whether or not the column can contain a null value
- the attributes of the column, for example if it is a primary key, or if it is to have an associated comment
- referential constraints on the table.

Generic clauses enable you to define extra column attributes for relational tables other than DB2 tables.
The DB2 facilities FIELDPROC, EDITPROC, VALIDPROC, AUDIT, COMMENT and LABEL are supported by the FIELDPROC, EDITPROC, VALIDPROC, AUDIT, DB2-COMMENT and DB2-LABEL clauses respectively.

You can specify the number of rows in a table using the CARDINALITY clause. This enables you to calculate the minimum and maximum size of the table.

DB2-TABLE repository definitions can be generated automatically if you use the Workbench Design Area (WBDA) facilities for DB2 database design.



AS

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

DB2-TABLE

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

CREATOR-OWNER
To define the owner of an object, enter:

```
CREATOR-OWNER user
```

user is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the object.

On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member. The CREATOR-OWNER clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in an export to DB2 panel.

Note: the DB2-USER or SQL-USER member named in the CREATOR-OWNER clause is used to generate user-qualified names for tables, views and aliases. If the DB2-USER has a LOCATION clause defined, and the DB2 profile is set to three-part name generation, location and user-qualified names are generated for tables and views. The location-qualifier can be overridden by a LOCATION clause defined in an export to DB2 panel.

DB2-COMMENT
To define a comment for an alias, table, view, or column, enter a string of no more than 254 characters.
When generated, lines of comment are concatenated to form a single string. To preserve spaces between words, insert a space after the last character of each continuing line.
For example, the DB2-TABLE named MANAGER-NUMBER has an owner of PERSONNEL and the following comment defined:

This table contains the Manager number of every manager in each department.

The following SQL statement can be generated:

```
COMMENT ON TABLE PERSONNEL.MANAGER_NUMBER IS 'This table cont
ains the Manager number of every manager in each department.'
```

In this example the word "contains" has been split due to the margins set in the DB2 profile.

This clause must be present for the successful generation of SQL COMMENT ON statements.

Note:
- for columns, the comment definition must follow the CONTAINS clause defining that column or group of columns
- for tables and views, the comment definition should precede the COLUMNS clause that defines the columns of the table.

DB2-LABEL
To define a label for an alias, table, view or column, enter a string of no more than 30 characters.
This clause must be present for the successful generation of SQL LABEL ON statements.

Note:
- for columns, the label definition must follow the CONTAINS clause defining that column or group of columns
- for tables and views, the label definition should precede the COLUMNS clause that defines the columns of the table.

IN
To define the table space in which the table is created, enter:

```
IN tbspace
```

tbspace is the name of a DB2-TBSPACE member.

On encoding the member specified is checked to ensure it is a DB2-TBSPACE member.

For the successful generation of SQL statements the DB2-TABLE definition must include an IN clause, naming a DB2-TBSPACE member. The DB2-TBSPACE must include an IN clause, naming a DB2-DATABASE. This chain of relationships defines the database to which the table space, and table belong.

COLUMNS
To specify which form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns in the table, enter:

```
COLUMNS form-keyword
```

form-keyword is one of the following:

```
ENTERED-AS
HELD-AS
REPORTED-AS
DEFAULTED-AS
```

The form keyword that you define in the COLUMNS clause applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.

For example, a DB2-TABLE member containing the following lines:

```
COLUMNS ENTERED-AS
CONTAINS ITEM1, ITEM2
```

refers to the two ITEM members:

ITEM1

```
ITEM
HELD-AS BINARY 10
ENTERED-AS CHAR 5
REPORTED-AS FLOAT 9
```

ITEM2

```
ITEM
ENTERED-AS DECIMAL 4.2
DEFAULTED-AS FLOAT 7
```

The ENTERED-AS form keyword in the DB2-TABLE definition specifies that the ENTERED-AS form description from both ITEMs is used to define the data type of columns. Therefore the column generated from ITEM1 has a data type of CHAR, and the column generated from ITEM2 has a data type of DECIMAL.
If you do not specify a form keyword then the DEFAULTED-AS form description is used. Where the ITEM or GROUP has no DEFAULTED-AS form description defined, then the MANAGER Products defaults apply. For further information refer to the DATAMANAGER Source Language Generation manual, section 2.2.3, remarks 39 to 41.

To specify the ITEM or GROUP members that define columns, enter:

```
CONTAINS member-list
```

member-list is the name of one or more ITEM or GROUP members, separated by commas, each representing a single column.

On encoding, the members specified in the CONTAINS clause are checked to ensure that they are either ITEMs or GROUPs. Duplicate column names are not permitted by DB2, therefore column names are checked on generation to ensure that no duplicates are present.

Each ITEM can define up to 15 form descriptions. To define which of the form descriptions you want to use, enter:

```
CONTAINS item version
```

item is the name of an ITEM member.

version is an integer in the range 1 to 15, and defines the form description version that you want to use.
For example:

```
HELD-AS CONTAINS STOCK-LIST 3
```

defines that the third HELD-AS form description in the ITEM member STOCK-LIST is used as the column data type.

To define a set of columns with identical attributes, using the same ITEM or GROUP member, enter:

```
CONTAINS (integer) member
```

integer is the number of columns to be derived from the member, within brackets.

member is the name of an ITEM or GROUP member.

To resolve duplicated names on generation of an SQL statement, column names are automatically suffixed by an underscore and a number, the first by_1, the second by_2 and so on.
For example:

```
CONTAINS (4) STOCK-LIST
```

generates the four columns STOCK_LIST_1, STOCK_LIST_2, STOCK_LIST_3 and STOCK_LIST_4. The generated attributes, such as data type, are the same for each of the four columns.
When a column is derived from a GROUP containing several ITEMs, the data type of the column is taken as CHAR. The maximum number of characters allowed in the column is calculated from the combined field lengths of the form descriptions defined in each ITEM. However, where a DB2 command applied to the DB2-TABLE specifies the EXPAND keyword, then each ITEM within a GROUP generates a separate column.

You can define additional attributes for a table's column(s) using sub-clauses within the CONTAINS clause. If the CONTAINS clause defines a column set or an expanded GROUP, the generated column attributes apply to all the columns in the set.

To specify the contents of the column as bit (binary) data, enter:

```
FOR-BIT-DATA
```

Note: the FOR-BIT-DATA keyword is only valid where columns have the data-type of CHAR, VARCHAR or LONG VARCHAR specified in the form description.

To define that columns cannot contain a null value, and are set to a default value by DB2 , enter:

```
WITH-DEFAULT
```

The WITH-DEFAULT keyword generates NOT NULL WITH DEFAULT in SQL statements.

To specify that a column cannot contain a null value, enter:

```
NOT-NULL
```

NOT-NULL and WITH-DEFAULT are mutually exclusive. If you specify both the member will not encode.

Columns with a data type of CHAR can also have an associated FIELDPROC procedure, provided WITH-DEFAULT is not defined. To define a FIELDPROC, enter:

```
FIELDPROC process
```

process is the name of a SYSTEM, PROGRAM or MODULE member, and represents a field procedure that exists in DB2. To define the parameters passed to the field procedure, enter:

```
CONSTANT 'list'
```

list is one or more parameters separated by commas. The list must be no more than 254 characters, within delimiters. For example:

```
CONTAINS IT-INCOMING NOT-NULL FIELDPROC MOD-XT3
CONSTANT "CONST4-3, CONST4-4"
```

defines that the ITEM member IT-INCOMING generates a column that:

- cannot contain a null value, and
- uses the process member MOD-XT3, passing the parameters CONST4-3 and CONST 4-4.

To define that a column forms the primary key of a table, enter:

PRIMARY-KEY
A primary key can comprise a maximum of 16 columns. For the successful generation of an SQL CREATE TABLE statement, columns defined as PRIMARY-KEY must also be defined as either NOT-NULL or WITH-DEFAULT.

Where your primary key consists of several columns, the sequence in which columns constitute the key is assumed to be the sequence in which they are defined, unless you specify the key position.

To define the position of the column in a multi-column primary-key, enter:

```
PRIMARY-KEY kpos
```

kpos is an integer in the range 1 to 16.

For example:

```
CONTAINS
  DEPT-NO
, EMP-NO NOT-NULL PRIMARY-KEY 2
, JOB-TITLE
, EMP-NAME NOT-NULL PRIMARY-KEY 1
```

defines that the primary key of this table is EMP-NAME followed by EMP-NO, rather than EMP-NO followed by EMP-NAME.
To define that the entries in a primary key column are sorted in ascending key order in the index, enter:

```
PRIMARY-KEY ASC
```

To define that the entries in a primary key column are sorted in descending key order in the index, enter:

```
PRIMARY-KEY DESC
```

To define that a column is to contain a comment enter:

```
DB2-COMMENT 'comment'
```

To define that a column is to contain a label enter:

```
DB2-LABEL 'label'
```

To define a referential constraint for a table, enter:

```
CONSTRAINT constraint
```

constraint is one, several or all of the following:

- the NAMED clause defines a constraint name
- the FOREIGN-KEY clause specifies one or more columns to form the foreign key
- the REFERENCES clause specifies the table being referenced
- the DELETE clause defines the associated DELETE rule.

You may define any number of referential constraints for a table. Each referential constraint requires its own CONSTRAINT clause. Each CONSTRAINT clause must include the FOREIGN-KEY and REFERENCES clauses for the successful generation of SQL CREATE statements.

You can name a referential constraint without specifying the column(s) that form the foreign key. Thus you can set up DB2-TABLE definitions with named referential constraints between them before deciding on the contents of the tables. This feature is useful in a top-down approach to database design. To define a constraint name, enter:

```
NAMED constraint-name
```

constraint-name is the name, of no more than 8 characters, by which the referential constraint is known to DB2.
Each constraint name must be unique within a table. If you do not specify a constraint name, a default name is generated by DB2.

To define the foreign key for a constraint, enter:

```
FOREIGN-KEY
```

followed by clauses that define the columns that comprise the foreign key, that is:

- specifying that one or more ITEMs and/or GROUPs each define a single column, see item 7
- specifying that each ITEM contained in a GROUP defines one column, see item 10.

The columns comprising the foreign key must already be defined in the CONTAINS clause, as the foreign key must exist as a column of the table.

You can name the foreign key column using the KNOWN-AS clause. When you generate an SQL statement the name defined in the KNOWN-AS clause is checked against the columns generated. The column name and foreign key name must be the same.
For example, the columns of a table are generated from an expanded GROUP member. However the foreign key comprises only one of the table's columns.

The DB2-TABLE definition includes the clauses:

```
CONTAINS MANY-ITEMS EXPAND
CONSTRAINT FOREIGN-KEY ITEM4 KNOWN-AS COL-4
```

Although the CONTAINS clause does not name the member ITEM4, the GROUP, MANY-ITEMS, includes the clauses:

```
CONTAINS
   ITEM1 KNOWN-AS COL-1
 , ITEM2 KNOWN-AS COL-2
 , ITEM3 KNOWN-AS COL-3
 , ITEM4 KNOWN-AS COL-4
```

The generated table has four columns, COL_1, COL_2, COL_3 and COL_4. The foreign key column is COL_4.
You can clarify correspondence between members in the repository where:

- foreign key column names differ from corresponding primary key column names
- different ITEM and/or GROUP members represent the foreign key columns in one table and the corresponding primary key columns in another table

except where the foreign key is defined using an expanded GROUP.

To specify the column(s) in another table to which the foreign key refers, enter:

```
MEMBER member
```

member is the name of an ITEM or GROUP member defining the column(s).

Where the name of a foreign key column(s) is different from its name in another table, you can optionally specify the local name of the column using the KNOWN-AS clause (see item 9).
To define the table to which a foreign key refers, enter:

```
REFERENCES table
```

table is the name of a DB2-TABLE or SQL-TABLE member.

To successfully generate SQL statements:

- one REFERENCES clause must be defined for each foreign key specified
- the referenced DB2-TABLE must exist and include a valid CREATOR-OWNER clause.

To define the delete rule for the constraint, enter:

```
DELETE option
```

option is RESTRICT, CASCADE or SET-NULL.
The keywords are similar to those used by DB2 and they have the same meanings.

If you do not define a DELETE rule the DB2 default applies.

LIKE
In DB2, you may wish to create a table with the same column structure as an existing table, so that you can use it in production and development. To copy the columns of an existing table in DB2, enter:

```
LIKE member
```

member is the name of a DB2-TABLE, SQL-TABLE, DB2-VIEW or SQL-VIEW member.

When you generate SQL CREATE TABLE statements from a DB2-TABLE that includes a LIKE clause, the statement also contains a LIKE clause. When the statement is submitted to DB2 the table or view named in the LIKE clause must exist, or the column structure cannot be copied.

If you want to generate SQL DECLARE statements or host language data structures from a DB2-TABLE defined using a LIKE clause, you must also define an AS clause referring to the same member. For example:

```
LIKE TA-TOTAL-STOCK
AS TA-TOTAL-STOCK
```

defines that the table copies the generated columns of the DB2-TABLE TA-TOTAL-STOCK.

EDITPROC
To define an edit routine for the table, enter:

```
EDITPROC process
```

process is the name of a SYSTEM, MMR-SYSTEM, PROGRAM or MODULE member.

The member represents an edit routine that must exist in DB2. In DB2, the edit routine is invoked whenever a row in the table is is retrieved, updated or inserted.

VALIDPROC
To define a validation routine for the table, enter:

```
VALIDPROC process
```

process is the name of a SYSTEM, MMRSYSTEM, PROGRAM or MODULE member.

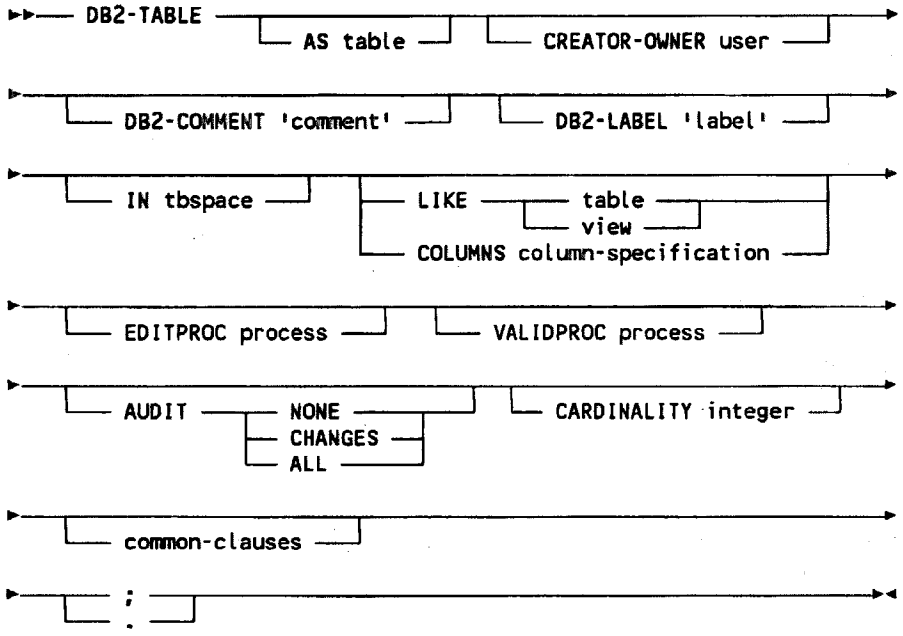
The member represents a validation routine that must exist in DB2. In DB2, the validation routine receives an entire table row as input and may be used to control a subsequent INSERT, UPDATE or DELETE statement.

AUDIT
The auditing options for the table. Valid values are NONE, CHANGES and ALL.

CARDINALITY
Specifies the approximate number of rows which will be contained in the table.

DB2-TABLE

Syntax



where

table is the name of a DB2-TABLE member

user is the name of a DB2-USER member

comment is a comment of no more than 254 characters

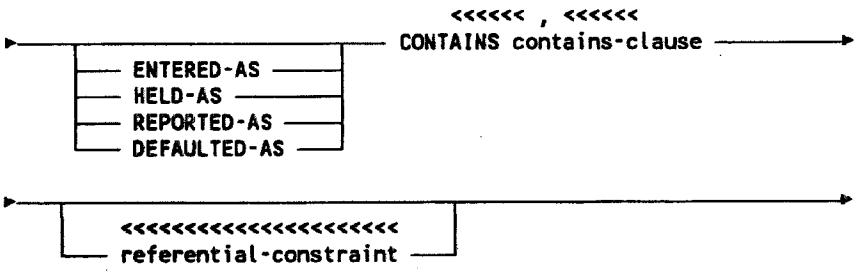
label is a label of no more than 30 characters

tbspace is the name of a DB2-TBSPACE member

table is as defined above

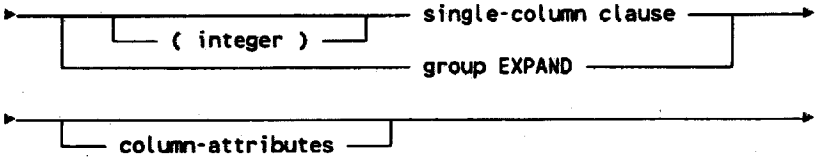
view is the name of a DB2-VIEW member

column-specification is:



where

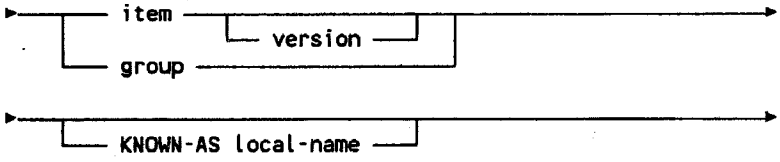
contains-clause is:



where

integer is the number of columns in a column set

single-column clause is:



where

item is the name of an ITEM member

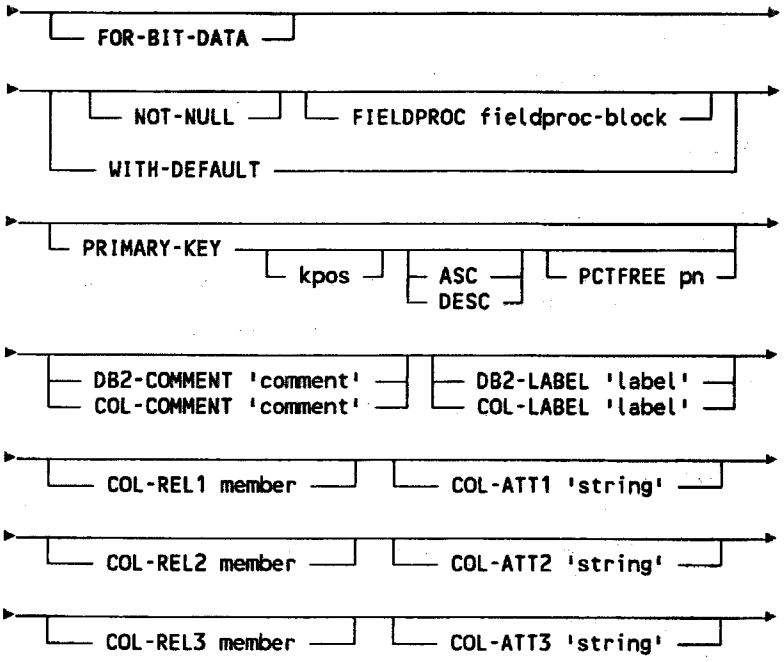
version is an integer in the range 1 to 15

group is the name of a GROUP member

local-name is the name of the column, consisting of no more than 18 characters.

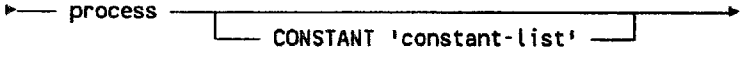
group is as defined above

column-attributes is:



where

fieldproc-block is:



where

process is the name of a SYSTEM, PROGRAM or MODULE member

constant-list is a character string of no more than 254 characters which contains one or more parameters; multiple parameters must be separated by commas.

kpos is an integer in the range 1 to 6, and represents the primary key position.

pn is an integer in the range 0 to 99

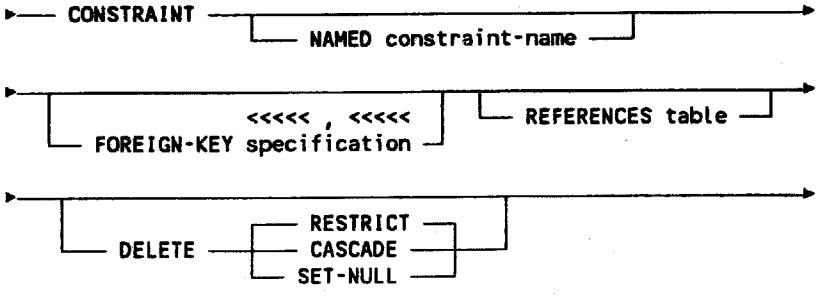
comment is as defined above

label is as defined above

member is the name of a repository member

string is a string of up to 254 characters.

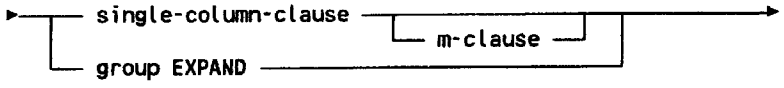
referential-constraint is:



where

constraint-name is the name of the constraint, consisting of no more than 8 characters

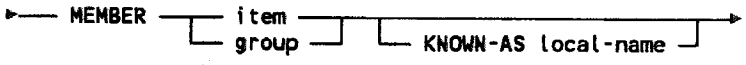
specification is:



where

single-column-clause is as defined above

m-clause is:



where

group and **item** are as defined above

local-name is as defined above.

group is as defined above.

process is as defined above

integer is an integer consisting of no more than 18 digits

common clauses are any of the clauses common to all member types.

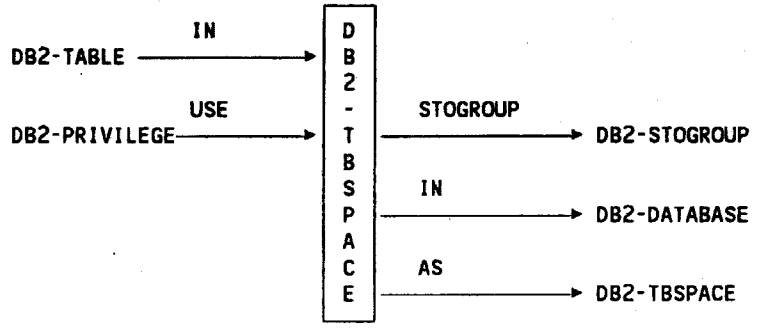
Refer to Appendix 2 for details of the common clauses.

DB2-TBSPACE

All the clauses available to define DB2-TBSPACE members are optional. However, the IN clause, defining the database to which the table space belongs, must be present for the successful generation of SQL CREATE TABLESPACE, DROP TABLESPACE or ALTER TABLESPACE statements.

In DB2, if a table space is partitioned, the corresponding index must also partitioned. Therefore for DB2-TBSPACE members defined with a PARTITION clause, you should define a corresponding clustered DB2-INDEX with the same partitions. **Note:** a DB2-TBSPACE member definition can contain either the PARTITION clause or the SEGSIZE clause. It cannot contain both, as a table space may not be partitioned and segmented at the same time.

The more commonly used relationships to and from DB2-TBSPACE are:



AS

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

DB2-TBSPACE

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

IN

To specify the database to which a table space belongs enter:

```
IN database
```

database is the name of a DB2-DATABASE member.

On encoding, the member specified in the IN clause is checked to ensure that it is a DB2-DATABASE member.
The IN clause must be defined to successfully generated SQL CREATE TABLESPACE and CREATE TABLE statements. The DB2-DATABASE member named in the IN clause is used to generate a database-qualified name for the table space.

STOGROUP
To define the physical space occupied by an index, table space or partition you can either:

- define the VSAM catalog it is to use

or:

- define the storage group it belongs to.

To define the VSAM catalog the index, table space or partition is to use, enter:

```
VCAT catalog
```

catalog is the name of a VSAM catalog, of no more than 8 characters.

To define the storage group to which the index, table space or partition belongs, enter:

```
STOGROUP stogroup-name
```

stogroup-name is the name of a DB2-STOGROUP member.

On encoding, the member specified in the STOGROUP clause is checked to ensure that it is a DB2-STOGROUP member.
The storage group defined in the DB2-DATABASE definition is the default used by indexes and table spaces:

- that belong to the database
- that do not have a storage group specified in their own member definition.

DB2-TBSPACE

When you define the STOGROUP clause, you can additionally define further details of primary and secondary storage using the PRIQTY, SECQTY sub-clauses. Both sub-clauses are expressed in kilobytes.

To specify the amount of primary storage space, enter:

```
PRIQTY p
```

p is the number of kilobytes, and must be in the range 3 to 4194304 inclusive.

To specify the amount of secondary storage space, enter:

```
SECQTY s
```

s is the number of kilobytes, and must be in the range 0 to 131068 inclusive.

Primary and secondary storage space is allocated in the storage area defined either:

- in the VOLUMES clause of the DB2-STOGROUP member named in the STOGROUP clause of the DB2-INDEX member

or:

- in the storage group defined in the STOGROUP clause of the DB2-DATABASE.

You can also specify whether or not DB2-defined data sets are to be erased when the index or partition is deleted in a DROP command.

If you want the data sets to be erased, enter:

```
ERASE YES
```

If you do not want the data sets to be erased, enter:

```
ERASE NO
```

If you do not specify PRIQTY, SECQTY or ERASE sub-clauses, the DB2 defaults apply.

VCAT
To define a control or master level password used to access the VSAM catalog, enter:

```
PASSWORD password
```

password is a control or master level VSAM catalog password, of no more than eight characters.
When you generate SQL statements, the VCAT password is taken directly from the password you specify in this clause.

FREEPAGE
You can accommodate future expansion of an index, table space or partition by defining the frequency with which pages are left free.

To define the relative frequency with which free pages are allocated, enter:

```
FREEPAGE fn
```

fn is an integer in the range 0 to 255.
For example, FREEPAGE 4 means that 1 free page is left after every 4 pages.

If you do not define a FREEPAGE clause the DB2 defaults apply.

PCTFREE
You can accommodate future expansion of an index, table space or partition by defining the percentage of each page that is left free.

To define the percentage space kept free on a page, when an index, table space or partition is loaded or reorganized, enter:

```
PCTFREE pn
```

pn is an integer in the range 0 to 99 .
If you do not define a PCTFREE clause the DB2 defaults apply.

DB2-TBSPACE

PARTITION
To define that an index or table space is to be partitioned, enter:

```
PARTITION
```

You can optionally define a number, details of storage, free space allocation and key values, for each partition.
To successfully generate SQL CREATE INDEX statements from DB2-INDEX members containing a PARTITION clause, you must define a NUMBER and KEY clause for each PARTITION.
To successfully generate SQL CREATE TABLESPACE statements from DB2-TBSPACE members containing a PARTITION clause, you must define a NUMBER clause for each PARTITION.

To give the partition a number, enter:

```
NUMBER n
```

n is an integer in the range 1 to 64. If you do not define the numbers of partitions, they are automatically generated by the DB2 CREATE command, commencing with 1, and increasing in increments of 1.

To define that the partitions of an index have a key value, enter:

```
KEY 'key-val'
```

key-val is the highest value, within delimiters, that a column in the partition can have.

If you are indexing more than one column, each key value must be enclosed in quotes and separated by a comma. The first key value corresponds to the first index column, the second value to the second index column and so on.

If the key value is a character field it must be enclosed in single quotes within double quotes. For example:

```
KEY " 'key-val' "
```

Note: you cannot define a key clause in DB2-TB-SPACE members. To define the storage space to which the partition belongs, use the VCAT, or STOGROUP, PRIQTY, SECQTY and ERASE clauses. To define how much space is left free in the partition, use the FREEPAGE and PCTFREE clauses.

If you do not define storage or free space for partitions individually, they automatically take space from that specified for the index or table space as a whole.

BUFFERPOOL
To define the buffer pool that table spaces use, enter:

```
BUFFERPOOL bufferpool-name
```

bufferpool-name is one of the following buffer pools:

```
BP0          BP1
BP2          BP32K
```

The buffer pool defined in the DB2-DATABASE definition is the default used by those table spaces that belong to the database, and do not have a buffer pool specified in their own member definition.

LOCKSIZE
To define the size of the storage unit that can be locked, enter:

```
LOCKSIZE unit
```

unit is one of the following units of locking:

```
ANY              PAGE
TABLESPACE       TABLE
```

If you do not specify a LOCKSIZE the DB2 default applies.

DB2-TBSPACE

CLOSE
To define that the data set, on which an index or table space resides, is to be closed when not in use, enter:

```
CLOSE YES
```

To define that it is to remain open, enter:

```
CLOSE NO
```

If you do not define a CLOSE clause the DB2 default applies.

DSETPASS
To define a password for the VSAM data set, on which an index or table space resides, enter:

```
DSETASS password
```

password is a VSAM data set password of no more than 8 characters.

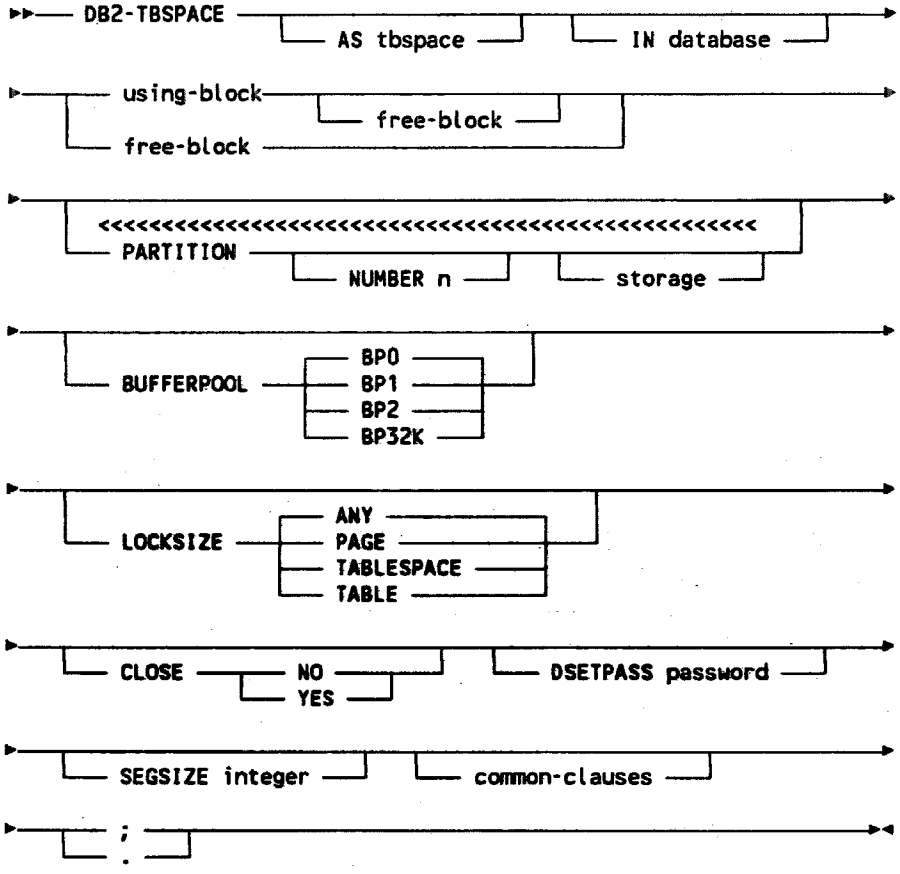
SEGSIZE
To define a segmented table space and the number of pages to be allocated to each segment, enter:

```
SEGSIZE integer
```

integer is a multiple of 4, in the range 4 to 64 inclusive.

Note: if you define a SEGSIZE clause you cannot define a PARTITION clause, as a segmented table space cannot be partitioned.

Syntax

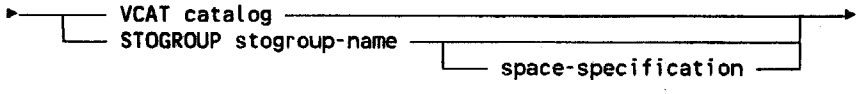


where

tbspace is the name of a DB2-TBSPACE member

database is the name of a DB2-DATABASE member

using-block is:

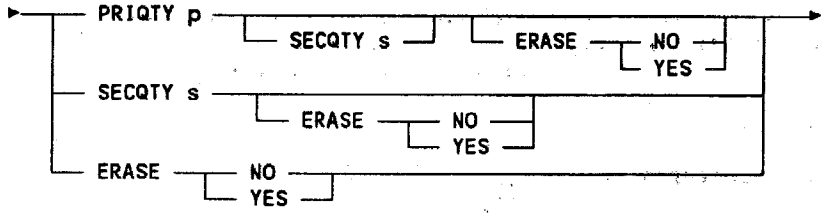


where

catalog-name is a VSAM catalog name of no more than 8 characters

stogroup-name is the name of a DB2-STOGROUP member

space-specification is:



where

p is an integer in the range 3 to 4194304

s is an integer in the range 0 to 131068.

free-block is:

where

fn is an integer in the range 0 to 255

pn is an integer in the range 0 to 99.

n is an integer in the range 1 to 64

storage is:

where

using-block is defined above

free-block is defined above.

password is a VSAM data set password, of no more than 8 characters

integer is an integer, which must be a multiple of 4, in the range 4 to 64

common clauses are any of the clauses common to all member types. Refer to Appendix 2 for details of the common clauses.

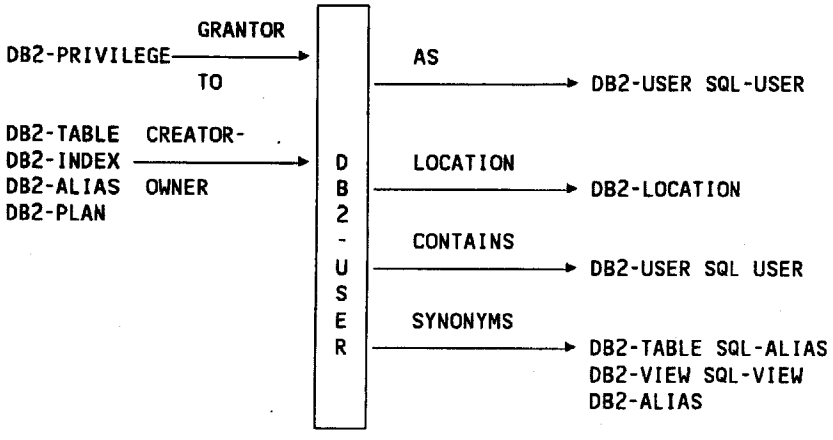
DB2-USER

In the DB2 environment all objects have an owner, who may also be the creator of the object. DB2-USER members document the owner or creator of DB2 objects.

DB2-USER members are specified as the recipients of privileges in DB2-PRIVILEGE members. DB2-USER members are also used to generate qualified names for DB2-ALIAS, DB2-INDEX, DB2-TABLE and DB2-VIEW members that have a CREATOR-OWNER clause defined. Note: the SQLID keyword in export to DB2 panels allows you to overrides the user name specified in the CREATOR-OWNER clause.

A DB2-USER member can be defined to represent a group of users, for example a project team. The DB2-USER member representing the group ID has a CONTAINS clause listing the DB2-USER members representing each member of the project team.

You can generate SQL CREATE SYNONYM statements from DB2-USER members if you define synonyms in the SYNONYMS clause. The more commonly used relationships to and from DB2-USER are:



AS
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

LOCATION
To define the location to which a database or user belongs, enter:

```
LOCATION location-name
```

location-name is the name of a DB2-LOCATION member that represents a local or remote location in a distributed network.

In DB2-USER member definitions, the LOCATION clause is used to generate location-qualified names for tables and views belonging to a particular user. The location name defined in a DB2-USER member can be overridden by a LOCATION clause in a DB2 command.

CONTAINS
To define a group ID, enter:

```
CONTAINS user-list
```

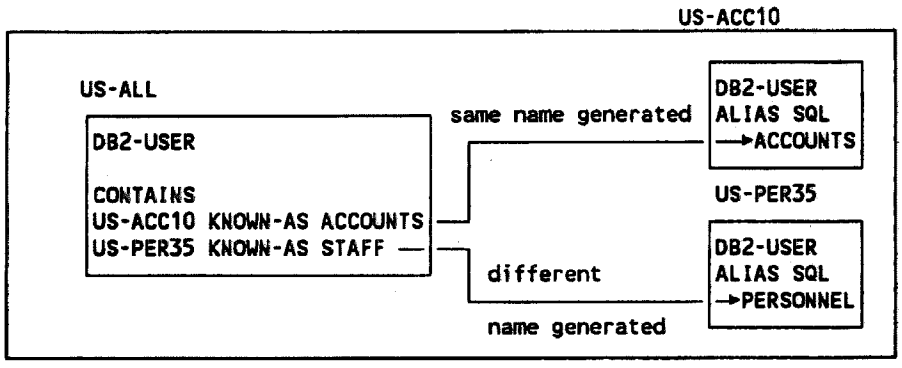
user-list is the name of one or more DB2-USER or SQL-USER members, separated by commas.

Each DB2-USER member represents one authorization ID, and together represent all the users who can use the group ID. DB2-USER members containing group IDs can be nested.
You can optionally define the owner name known by DB2, where it is different from the member name.
To define the DB2 owner name, enter:

```
KNOWN-AS local-name
```

local-name is a creator or owner name, of no more than eight characters, recognized by DB2.

To prevent conflicting definitions, the local name specified in the KNOWN-AS clause should be the same as any SQL ALIAS defined in the DB2-USER to which it refers. For example, in the diagram below the member US-ALL contains two other DB2-USER members, US-ACC10 and US-PER35.



When you generate SQL GRANT or REVOKE statements from US-ALL, the KNOWN-AS clause gives owner names of ACCOUNTS and STAFF. However when you generate SQL GRANT or REVOKE statements directly from US-PER35, the SQL ALIAS gives a different owner name of PERSONNEL.

If the DB2-USER member, named in the TO clause of a DB2-PRIVILEGE, is a group ID, generated GRANT or REVOKE statements only name the group ID. If the group ID is also defined in the DB2 environment, all the authorization IDs in the group inherit the privileges granted to the group ID, and lose privileges when they are revoked.

However, if you use the USER-EXPANSION keywords available in the DB2 GRANT and DB2 REVOKE commands, the group ID is expanded so that the privilege either applies to the group ID and all the individual user IDs within it, or applies only to the individual user IDs.

SYNONYMS

To define synonyms for the aliases, tables and views of a particular user, enter:

```
SYNONYMS synonym-name FOR object
```

synonym-name is an alternative name, of no more than eighteen characters, for the alias, table or view.

object is the name of a DB2-ALIAS, DB2-TABLE, SQL-TABLE, DB2-VIEW or SQL-VIEW member.

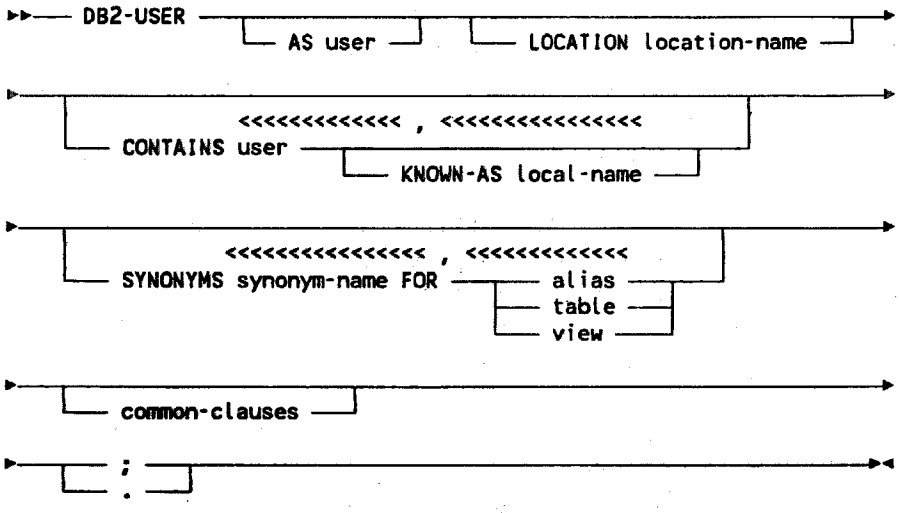
DB2-USER

The DB2 synonym for the alias, table or view is taken directly from the name you specify in this clause.

To define several synonyms, each synonym and the object it defines must be separated by commas. For example:

```
SYNONYMS EMP-CODES FOR TA-EMP7C
       , TAX-RECS FOR VW-TAX9N
```

Syntax



where

user is the name of a DB2-USER or SQL-USER member

location-name is the name of a DB2-LOCATION member

user is as defined above

local-name is a local name, of no more than 18 characters

synonym-name is a synonym, of no more than 18 characters

alias is the name of a DB2-ALIAS member

table is the name of a DB2-TABLE or SQL-TABLE member

view is the name of a DB2-VIEW or SQL-VIEW member

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DB2-VIEW

Views are of major interest to the end-user, since they provide access to specific data, selected from one or more tables or views. Therefore, with DB2-TABLE, the DB2-VIEW is one of the most used DB2 member types.

All other clauses available to define DB2-VIEW members are optional. However, for the successful generation of SQL statements you must define specific clauses, as follows:

- for CREATE VIEW statements define the CREATOR-OWNER and COLUMNS clauses
- for COMMENT ON statements define the CREATOR-OWNER and DB2-COMMENT clauses
- for DECLARE VIEW statements define the CREATOR-OWNER and COLUMNS clauses
- for DROP VIEW statements define the CREATOR-OWNER clause
- for LABEL ON statements define the CREATOR-OWNER and DB2-LABEL clauses.

To specify the ITEM and GROUP members that represent the columns of the view, use the CONTAINS clause. As for DB2-TABLE members, it establishes relationships between a DB2-VIEW and the ITEM and GROUP members that define the view's columns. You can define columns:

- individually, so that one ITEM or GROUP member defines one column
- in sets, so that the same ITEM or GROUP member defines several columns, with identical attributes
- in cascades from a GROUP member, so that every ITEM nested in a GROUP member defines one column.

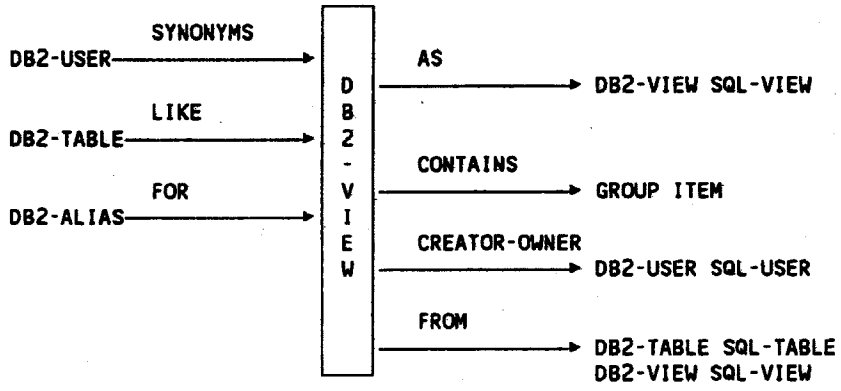
Sub-clauses within the CONTAINS clause enable you to define:

- the names of columns
- calculated values to be held by columns
- that columns are GROUP BY columns

- the table that the columns are derived from
- associated column comments and labels.

The DB2-VIEW member type has specific clauses that enable you to define the tables or views on which the view is based and to define a SELECT clause with optional FROM, WHERE and HAVING clauses.

DB2-VIEW repository definitions can be generated automatically if you use the Workbench Design Area facilities (WBDA) for DB2 database design.
The more commonly used relationships to and from DB2-VIEW are:



AS
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

DB2-VIEW

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

CREATOR-OWNER
To define the owner of an object, enter:

```
CREATOR-OWNER user
```

user is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the object.
On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member. The CREATOR-OWNER clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in an export to DB2 panel.

Note: the DB2-USER or SQL-USER member named in the CREATOR-OWNER clause is used to generate user-qualified names for tables, views and aliases. If the DB2-USER has a LOCATION clause defined, and the DB2 profile is set to three-part name generation, location and user-qualified names are generated for tables and views. The location-qualifier can be overridden by a LOCATION clause defined in an export to DB2 panel.

DB2-COMMENT
To define a comment for an alias, table, view, or column, enter a string of no more than 254 characters.
When generated, lines of comment are concatenated to form a single string. To preserve spaces between words, insert a space after the last character of each continuing line.
For example, the DB2-TABLE named MANAGER-NUMBER has an owner of PERSONNEL and the following comment defined:

This table contains the Manager number of every manager in each department.

The following SQL statement can be generated:

```
COMMENT ON TABLE PERSONNEL.MANAGER_NUMBER IS 'This table cont
ains the Manager number of every manager in each department'
```

In this example the word "contains" has been split due to the margins set in the DB2 profile.
This clause must be present for the successful generation of SQL COMMENT ON statements.

Note:

- for columns, the comment definition must follow the CONTAINS clause defining that column or group of columns
- for tables and views, the comment definition should precede the COLUMNS clause that defines the columns of the table.

DB2-LABEL
To define a label for an alias, table, view or column, enter a string of no more than 30 characters.
This clause must be present for the successful generation of SQL LABEL ON statements.

Note:
- for columns, the label definition must follow the CONTAINS clause defining that column or group of columns
- for tables and views, the label definition should precede the COLUMNS clause that defines the columns of the table.

DEFAULTED-AS
To specify that the DEFAULTED-AS form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns, enter:

```
DEFAULTED-AS
```

and delete:

```
ENTERED-AS
HELD-AS
REPORTED-AS
```

The form keyword that you define applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.
If you do not specify a form keyword then the DEFAULTED-AS form description is used.

HELD-AS
To specify that the HELD-AS form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns, enter:

```
HELD-AS
```

and delete:

```
ENTERED-AS
REPORTED-AS
DEFAULTED-AS
```

The form keyword that you define applies to$all the ITEMs and GROUPs named in the CONTAINS clause that follows.

If you do not specify a form keyword then the DEFAULTED-AS form description is used.

ENTERED-AS
To specify that the ENTERED-AS form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns, enter:

```
ENTERED-AS
```

and delete:

```
HELD-AS
REPORTED-AS
DEFAULTED-AS
```

The form keyword that you define applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.
For example, a DB2-TABLE, DB2-INDEX or DB2-VIEW member containing the following lines:

```
ENTERED-AS
CONTAINS ITEM1, ITEM2
```

refers to the two ITEM members:

ITEM1

```
ITEM
HELD-AS BINARY 10
ENTERED-AS CHAR 5
REPORTED-AS FLOAT 9
```

ITEM2

```
ITEM
ENTERED-AS DECIMAL 4.2
DEFAULTED-AS FLOAT 7
```

The ENTERED-AS form keyword in the DB2-INDEX or DB2-VIEW definition specifies that the ENTERED-AS form description from both ITEMs is used to define the data type of columns.
Therefore the column generated from ITEM1 has a data type of CHAR and the column generated from ITEM2 has a data type of DECIMAL.

If you do not specify a form keyword then the DEFAULTED-AS form description is used. Where the ITEM or GROUP has no DEFAULTED-AS form description defined, then the MANAGER Products defaults apply. For further details refer to the DATAMANAGER Source Language Generation manual, section 2.2.3 remarks 39 to 41.

REPORTED-AS
To specify that the REPORTED-AS form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns, enter:

```
REPORTED-AS
```

and delete:

```
HELD-AS
ENTERED-AS
DEFAULTED-AS
```

The form keyword that you define applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.
If you do not specify a form keyword then the DEFAULTED-AS form description is used.

CONTAINS
You can define additional attributes for a view's column(s) using sub-clauses within the CONTAINS clause. If the CONTAINS clause defines a column set or an expanded GROUP, the generated column attributes apply to all the columns in the set.
To define the name of a column in a view, if it is different form the column name in the table, enter:

```
COLUMN-NAME name
```

name is a name of no more than 18 characters. If you do not define a name for the column, the usual rules for generating column names apply.

If you define a COLUMN-NAME clause for a column set or expanded GROUP, the generated column names are identical, and the statement generation will fail.

To define a calculated value that is held in a column, enter:

```
EXPRESSION string
```

string is an SQL expression, of no more than 255 characters within delimiters, and contains the expression used to calculate the values contained in the column.

When you define an EXPRESSION clause you can generate a column that holds a value calculated by an operation performed on none, one or more of the other columns in the view.
The ITEM that defines the column should generate a data type compatible with the calculated expression, for SQL DECLARE or PRODUCE statements to be meaningful.

To define that a column is part of a GROUP BY clause for the view, enter:

```
GROUP-BY
```

The keyword has the same meaning as in DB2.
Ambiguity may arise if different columns from different tables or views have the same name. To define a correlation name for a column, so that it can be correlated with the table named in the FROM clause, enter:

```
TABLE correlation-name
```

correlation-name is an identifier, of no more than 18 characters, for the table or view. The same name is repeated in the CORRELATION-NAME clause, where it is associated with the member name of the DB2-TABLE.
For example:

```
CONTAINS
   IT-PERS-13  KNOWN-AS CODE-NAME   TABLE PERSON
 , IT-PROJ-58  KNOWN-AS CODE-NAME   TABLE PROJECT
FROM
   TA-EMP-DETAILS   CORRELATION-NAME PERSON
 , TA-PROJ-DATA     CORRELATION-NAME PROJECT
```

defines that:

- the ITEM member IT-PERS-13 generates a column called CODE-NAME
- the ITEM member IT-PROJ-58 generates a column called CODE-NAME
- columns with a correlation-name of PERSON in the TABLE clause are derived from the DB2-TABLE TA-EMP-DETAILS
- columns with the correlation name of PROJECT are derived from the DB2-TABLE TA-PROJ-DATA.

To define that a column is to contain a comment enter:

```
DB2-COMMENT comment
```

To define that a column is to contain a label enter:

```
DB2-LABEL label
```

WITH-CHECK-OPTION
To define that DB2 checks any updates to a view against the view definition, enter:

```
WITH-CHECK-OPTION
```

SELECT
To define that duplicate rows in a view are all preserved, enter:

```
SELECT ALL
```

To define duplicates are eliminated, enter:

```
SELECT DISTINCT
```

If you do not define a SELECT clause, the SELECT attribute is automatically generated, and the DB2 default applies.

FROM
To define the tables or views upon which a view is based, enter:

```
FROM member
```

member is the name of a DB2-TABLE, SQL-TABLE, DB2-VIEW or SQL-VIEW member.

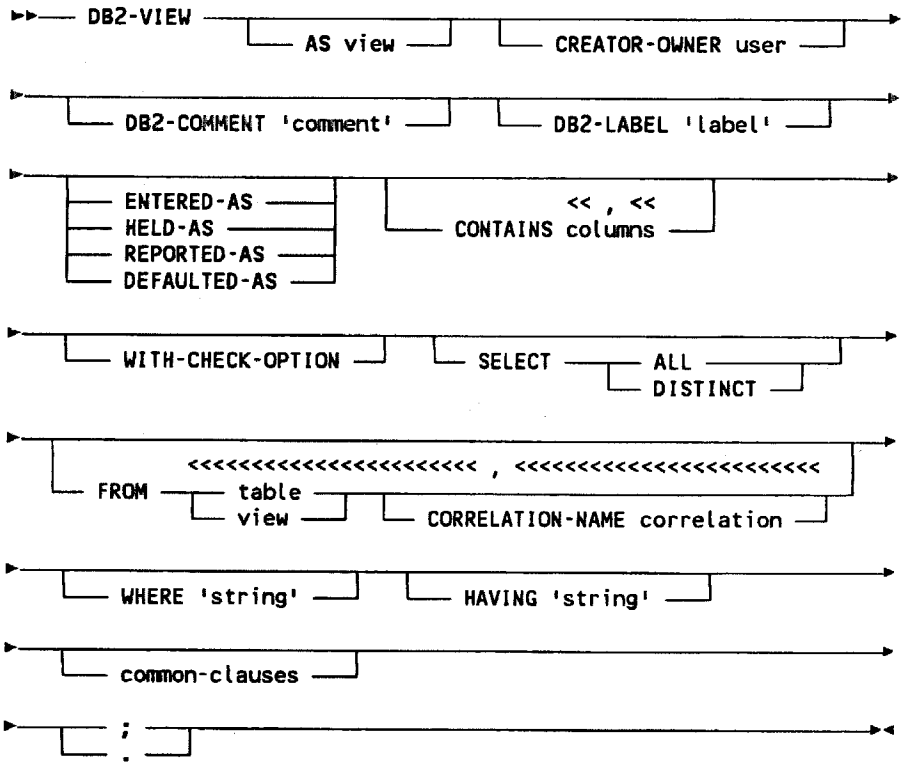
For the successful generation of SQL CREATE statements the members named in the FROM clause must include a valid CREATOR-OWNER clause.

WHERE
The WHERE sub-clause for the view's subselect. Note that the clause is NOT checked for correct SQL syntax.

HAVING
The HAVING sub-clause for the view's subselect. Note that the clause is NOT checked for correct SQL syntax.

DB2-VIEW

Syntax



where

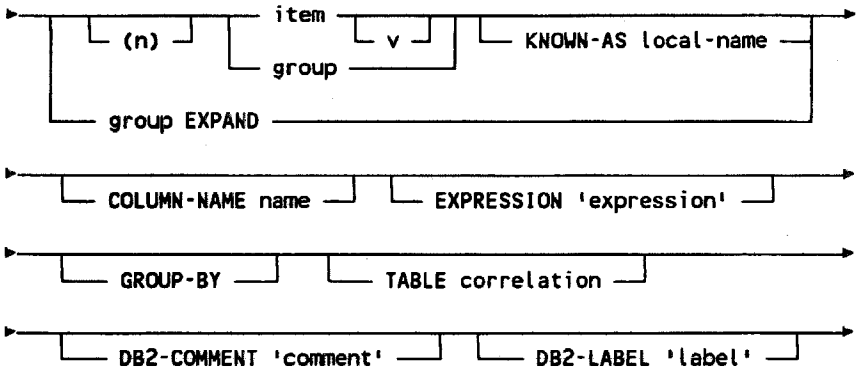
view is the name of a DB2-VIEW member

user is the name of a DB2-USER member

comment is a character string of no more than 254 characters, within delimiters

label is a character string of no more than 30 characters, within delimiters

columns is:



where

(n) is the number of columns in a column set

item is the name of an ITEM member

v is an integer in the range 1 to 15

group is the name of a GROUP member

local-name is the name of the column in the table and consists of no more than 18 characters

name is the name of the column in the view and consists of no more than 18 characters

expression is an expression of no more than 255 characters, within delimiters

correlation is the name of a correlation and consists of no more than 18 characters

comment is defined above

label is defined above.

DB2-VIEW

table is the name of a DB2-TABLE member

view is defined above

correlation is defined above

string is a valid SQL search condition

string is as defined above

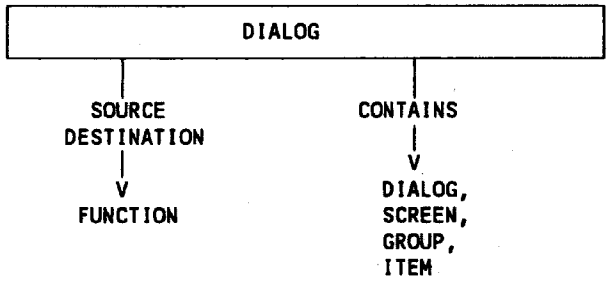
common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DIALOG

A DIALOG is a series of interactive exchanges; each exchange comprises a display of information by the system followed by a user response, as defined by one or more functions.

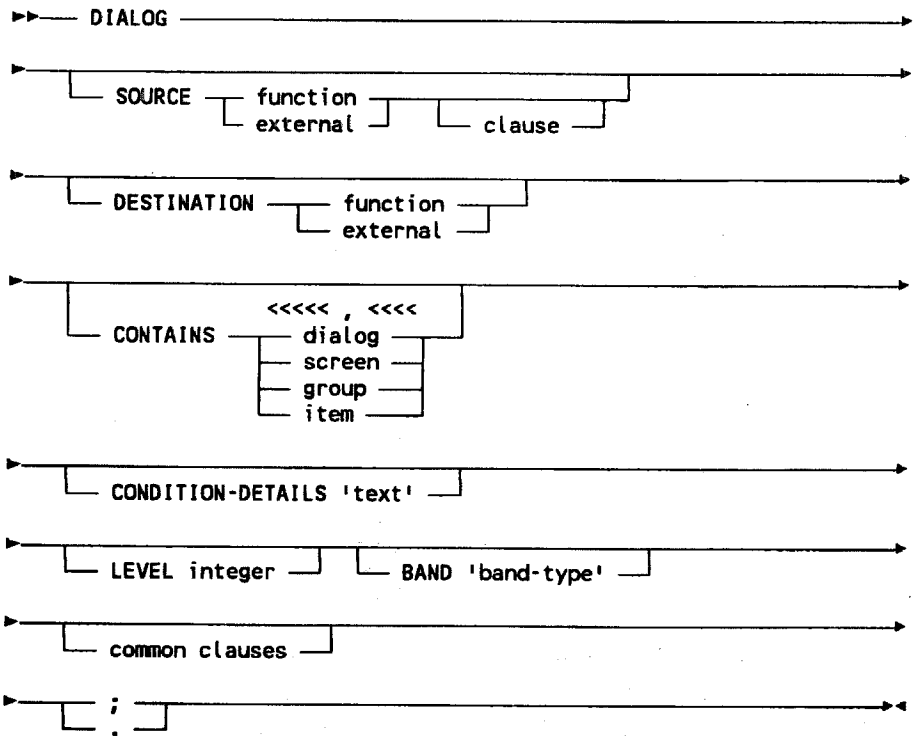
Valid relationships to and from the DIALOG member type are shown in the following diagram:



A DIALOG member definition must begin with the member identifier keyword DIALOG. The clauses and keywords forming the body of the member definition statement are all optional. They can be specified in any order, except for the CONTAINS clause which must not precede the form-keyword, or be followed by an alignment keyword.

DIALOG

Syntax

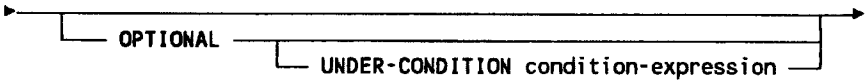


where

function is the name of a FUNCTION member

external is the name of an EXTERNAL member

clause is:



where **condition-expression** is a string of up to 79 characters, including delimiters

dialog is the name of a DIALOG member

screen is the name of a SCREEN member

group is the name of a GROUP member

item is the name of an ITEM member

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters

integer is an integer value of up to 18 digits, optionally preceded by a sign

' band-type' is a text string of up to 78 characters, including delimiters.

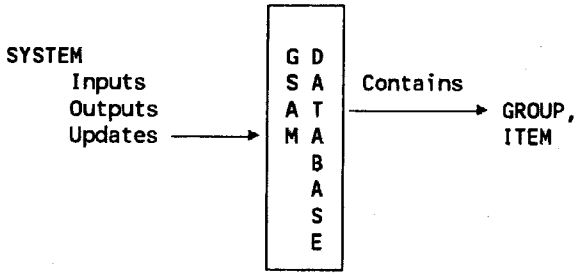
Note: the delimiters shown in the above syntax are required when defining a DIALOG member via the command interface.

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

DL1-GSAM-DATABASE

A GSAM DATABASE allows the allocation of a file to the IMS system. The more commonly used relationships to and from GSAM DATABASE are:



For further details refer to the IMS (DL/1) Interface documentation.

ACCESS
Defines the access method for the database dataset(s), which may be OSAM or VSAM for H/D databases,
ISAM or VSAM for HISAM databases,
and VSAM or BSAM for GSAM databases.
PASSWORD MAY ALSO BE SPECIFIED FOR PASSWORD PROTECTION.

DATASETS
Details of the input and output files for a GSAM database.

HELD-AS
This clause is used if the member contains items and groups in the HELD-AS form. It is an alternative to ENTERED-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

ENTERED-AS
This clause is used if the member contains items and groups in the ENTERED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

REPORTED-AS
This clause is used if the member contains items and groups in the REPORTED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

DEFAULTED-AS
This clause is used if the member contains items and groups in the DEFAULTED-AS form. It is an alternative to HELD-AS, REPORTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

ALIGNED
This clause is used if all the members in the CONTAINS clause are ALIGNED. It is an alternative to UNALIGNED or NOT-ALIGNED. Only one such clause is valid in the definition.

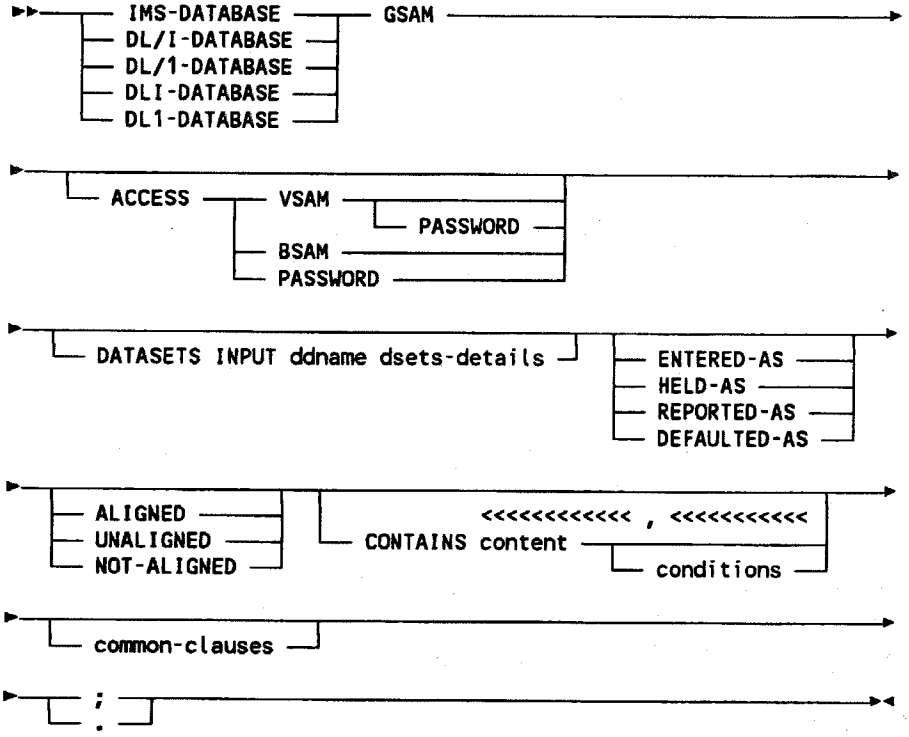
UNALIGNED
This clause is used if all the members in the CONTAINS clause are UNALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

NOT-ALIGNED
This clause is used if all the members in the CONTAINS clause are NOT-ALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

CONTAINS
Enter groups, items etc. contained by GSAM d/b.

DL1-GSAM-DATABASE

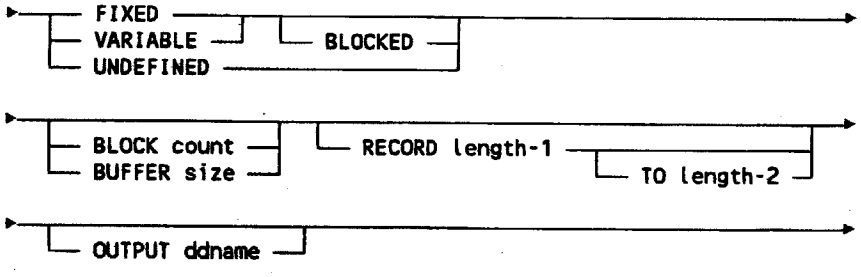
Syntax



where

ddname is 1 to 8 alphanumeric characters, being the logical name used in the job control to identify the database data set.

dsets-details are:



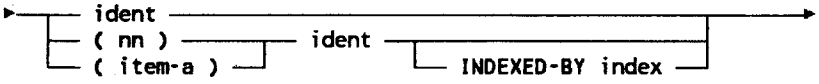
where

count is an unsigned, non-zero integer, being the number of logical records per physical block

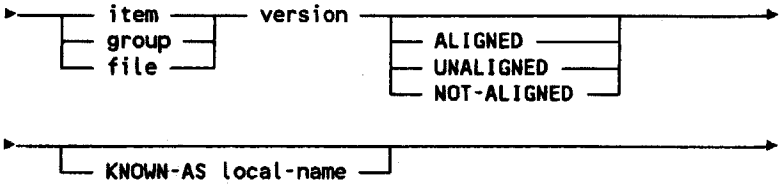
size is an unsigned, non-zero integer, being the number of bytes per physical block or control interval

length-1, **length-2** are non-zero integers

content is:



where **ident** is:



where:

item is the name of an ITEM member

group is the name of a GROUP member

file is the name of a FILE member

version is an unsigned integer in the range 1 to 15

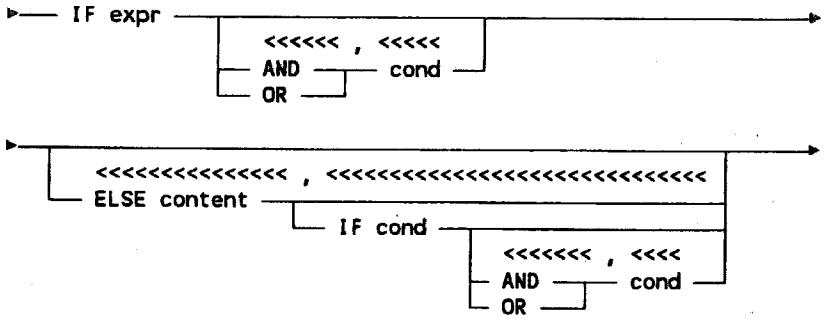
local-name is a name, conforming to the rules for member names

nn is an unsigned integer of from 1 to 18 digits, being the number of times **item** or **group** occurs in the array

item-a is the name of an ITEM.

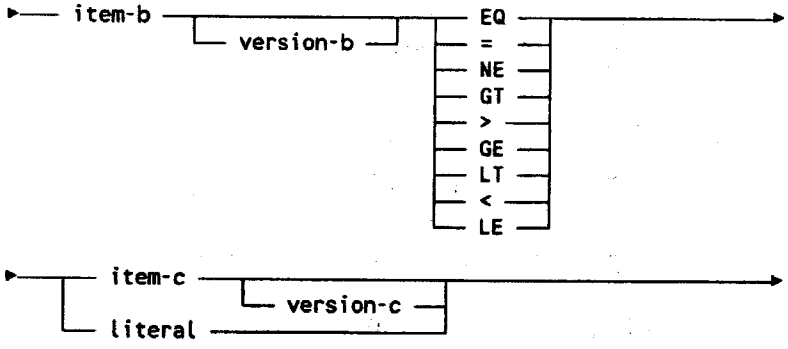
index is a name, conforming to the rules for member names, that is to be used as the index name when COBOL data descriptions are generated by the Source Language Generation facility

conditions are:



where

cond is:



where:

literal is a literal comparand

item-b is the name of the ITEM whose contents are to be compared with the comparand

version-b is an unsigned integer in the range 1 to 15

item-c is the name of the ITEM whose contents are the comparand

version-c is an unsigned integer in the range 1 to 15

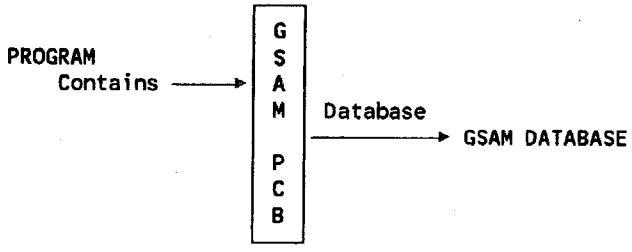
common-clauses are any of the clauses common to all member types.

Note: the commas and delimiters shown in the above syntax are required when defining a DL1-GSAM-DATABASE member via the command interface.

Refer to Appendix 2 for details of the common clauses.

DL1-GSAM-PCB

A GSAM PCB defines the dataset allocation for a GSAM database or file processing program communication block (PCB).
The more commonly used relationships to and from GSAM PCB are:



For further details refer to the "IMS (DL/1) Interface" documentation.

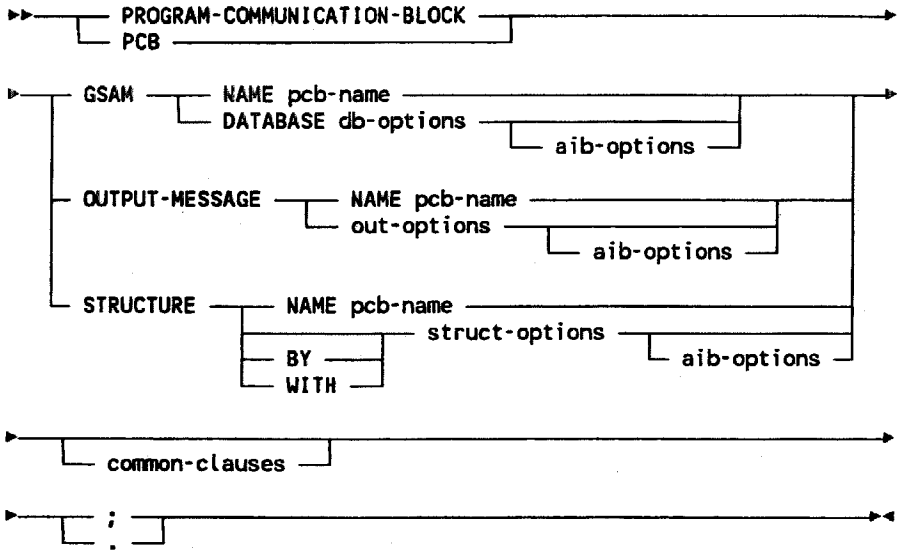
NAME
Name another PCB upon which the attributes of this PCB will be based. If specified then only common clauses should follow.

DATABASE
Name of d/b to be processed by a GSAM PCB.

BY
Processing options for a GSAM PCB such as LOAD or GET.

AIB-LIST-ADDRESS
Enter YES or NO to signify whether this PCB address is required in the application interface-control block for IMS releases from 2.0. Will cause generation of LIST = YES/NO when the PRODUCE IMS command should also specify OPTIONS including PCB-NAME.

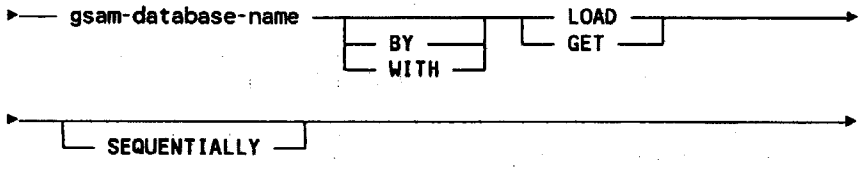
Syntax



where

pcb-name is the name of another PROGRAM-COMMUNICATION-BLOCK member

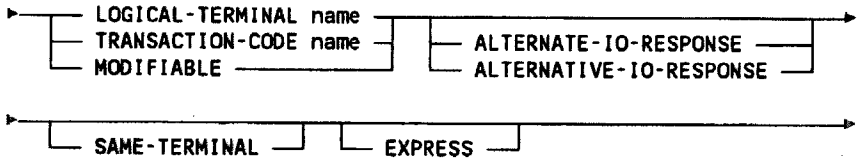
db-options are:



where **gsam-database-name** is the name of a database member of the GSAM type.

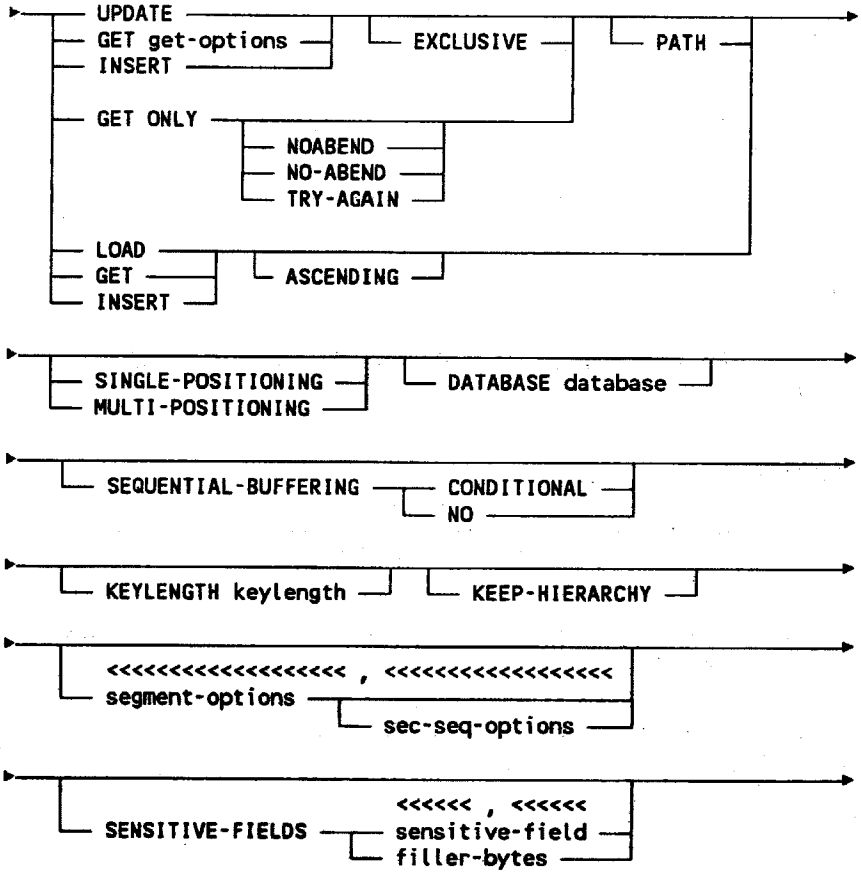
aib-options are:

out-options are:



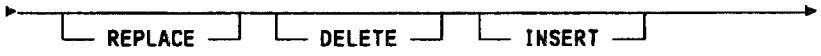
where **name** is an alphanumeric name 1 to 8 characters in length.

structure-options are:



where

get-options are:



Note: The UPDATE, GET, INSERT, LOAD, EXCLUSIVE, ASCENDING, PATH, GET ONLY, NOABEND, TRY-AGAIN, REPLACE and DELETE keywords can all be optionally separated by commas.

database is the name of a DATABASE member

keylength is an integer in the range 0 to 32767

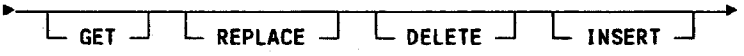
segment-options are:

where

segment is the name of a SEGMENT member

struct-options-2 are:

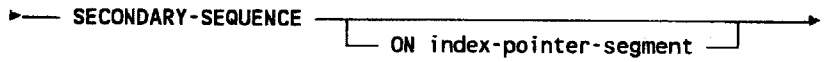
where **grdi-options** are:



Note 1: The UPDATE, KEY-SENSITIVE, GET, REPLACE, DELETE, INSERT, EXCLUSIVE and PATH keywords can all be optionally separated by commas.

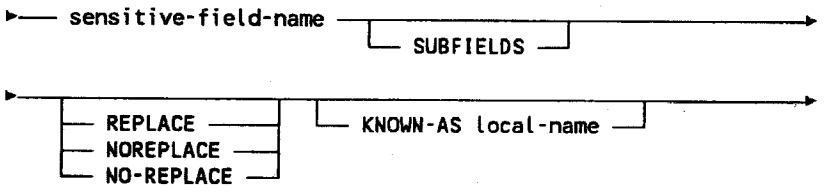
Note 2: You must specify at least one keyword in **grdi-options**.

sec-seq-options are:



where **index-pointer-segment** is the name of an INDEX-POINTER SEGMENT member.

sensitive-field is:



where:

local-name is a name, conforming to the rules for member names stated in the CONTROLMANAGER User's Guide

sensitive-field-name is the name of a GROUP, ITEM or sequence key member or concatenated key member

filler-bytes is an unsigned integer in the range 1 to 32767

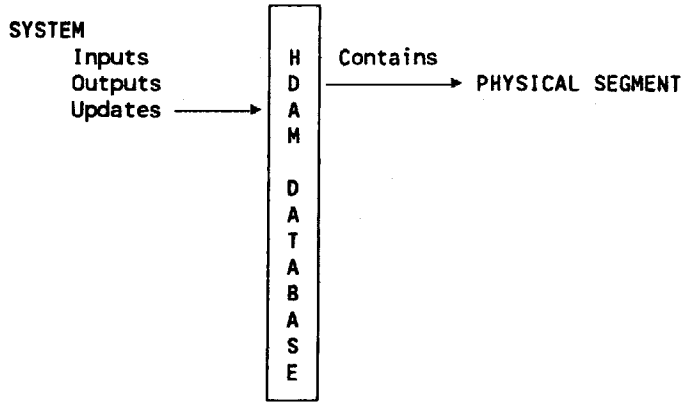
common-clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DL1-HDAM-DATABASE

A Hierarchical Direct Access Method (HDAM) DATABASE allows the storage of IMS segments for random retrieval.

The more commonly used relationships to and from HDAM PHYSICAL DATABASE are:



For further details refer to the "IMS (DL/1) Interface" documentation.

ACCESS
Defines the access method for the database dataset(s), which may be
OSAM or VSAM for H/D databases,
ISAM or VSAM for HISAM databases,
and VSAM or BSAM for GSAM databases.
PASSWORD MAY ALSO BE SPECIFIED FOR PASSWORD PROTECTION.

RANDOMIZING-MODULE
Defines the randomiser

ANCHOR-POINTS
Defines the number of RAPS/block

RELATIVE-BLOCK-MAXIMUM
Defines the number of blocks in the RAA

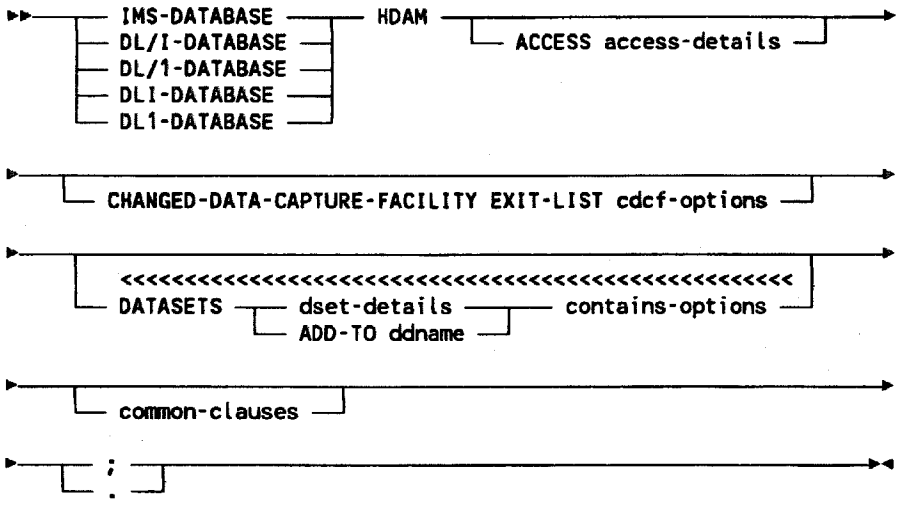
DL1-HDAM-DATABASE

INSERTION-BYTES-MAXIMUM
Defines the byte limit for RAA insertion

CHANGED-DATA-CAPTURE-FACILITY
Defines the DCDCF exits with any options and the d/b version string

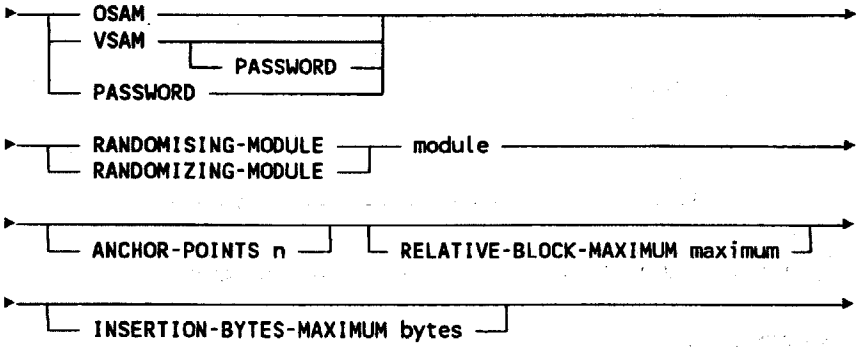
DATASETS
Enter details of the database dataset group contents.
BLOCK and BUFFER are alternatives when specifying a PRIME dataset group. Use the ADD-TO option for continuation of a secondary dataset group segment contents list after specifying another PRIME dataset.
When ADD-TO is specified no physical blocking information applies.

Syntax



where:

access-details are:



where:

module is the name of a MODULE member

n is an unsigned integer in the range 1 to 255, being the number of root anchor points required in each control interval or block

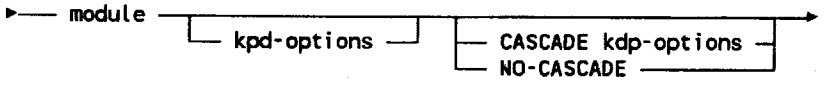
maximum is an unsigned integer in the range 1 to 16777215, being the maximum block number to be produced by the randomizing module

bytes is an unsigned integer in the range 1 to 16777215, being the maximum number of bytes to be inserted into the root addressable area.

cdcf-options are:

where:

module-list is:



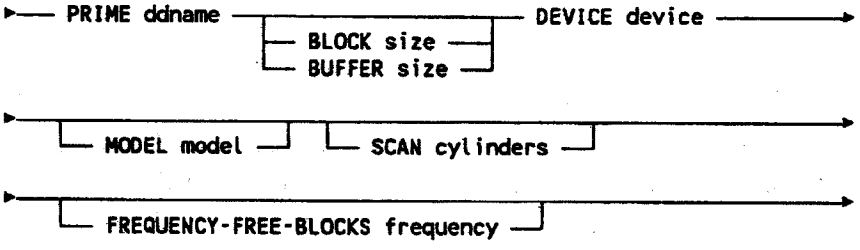
where:

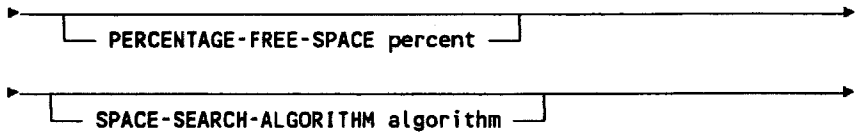
module is the name of a MODULE or PROGRAM member

kpd-options are:

db-version is a delimited string of up to 255 characters.

dsets-details are:





where:

ddname is 1 to 8 alphanumeric characters, being the logical name used in the job control to identify the physical file

size is the number of bytes required per physical block or control interval

device is one of the keywords or numbers from the list:

```
DRUM 2311 3310 3350
CELL 2314 3330 3370
2301 2319 3340 3375
2305 2321 3344 3380
3390
```

model is an integer, 1 or 2 if device is 2305, or 1 or 11 if device is 3330

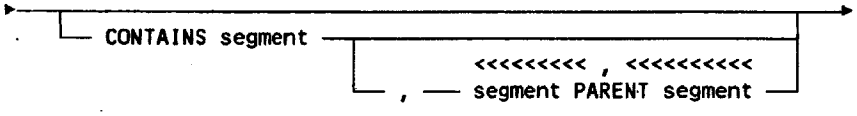
cylinders is an unsigned integer in the range 0 to 255

frequency is an unsigned integer in the range 2 to 100, or is 0

percent is an unsigned integer in the range 0 to 99

algorithm is an unsigned integer in the range 0 to 2.

contains-options are:



where **segment** is the name of any physical segment.

DL1-HDAM-DATABASE

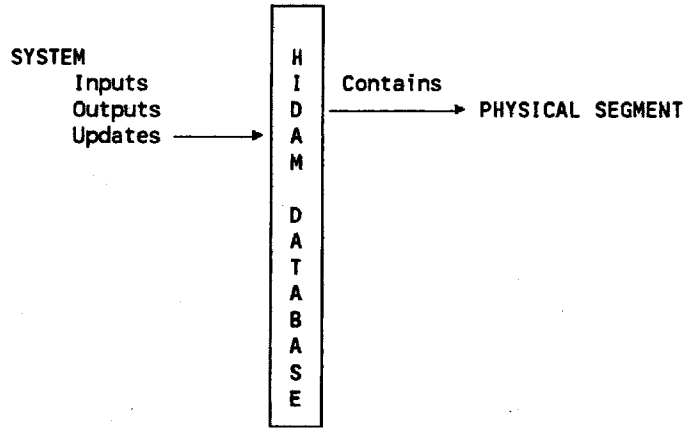
ddname is as defined above

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

DL1-HIDAM-DATABASE

A Hierarchical Indexed Direct Access Method (HIDAM) DATABASE allows the storage of IMS segments for both random and sequential retrieval. The more commonly used relationships to and from HIDAM PHYSICAL DATABASE are:



For further details refer to the "IMS (DL/1) Interface" documentation.

ACCESS
Defines the access method for the database dataset(s), which may be OSAM or VSAM for H/D databases,
ISAM or VSAM for HISAM databases,
and VSAM or BSAM for GSAM databases.
PASSWORD MAY ALSO BE SPECIFIED FOR PASSWORD PROTECTION.

INDEX
The index access method should be specified here
as ISAM or VSAM, for which DOS-COMPATIBLE and PASSWORD are options.

DATABASE
Defines the name of the primary index database

DL1-HIDAM-DATABASE

SEGMENT
Defines the index segment

SEQUENCE-KEY
Defines the index key field name

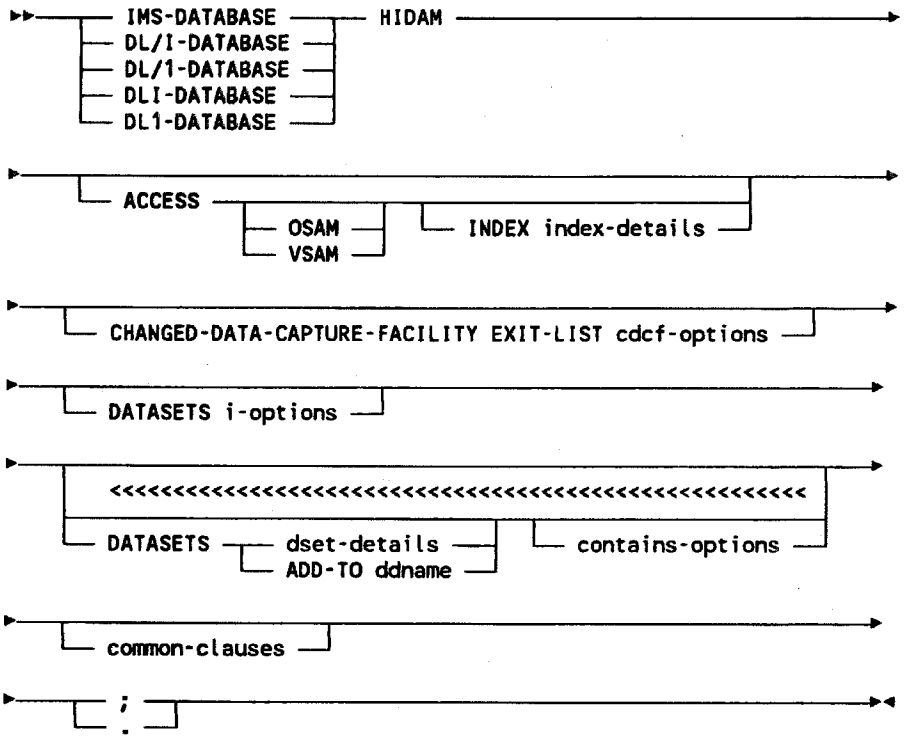
CHANGED-DATA-CAPTURE-FACILITY
Defines the DCDCF exits with any options and the d/b version string

DATASETS INDEX
Specify the HIDAM primary index dataset allocation details.
OVERFLOW dataset details may be added prior to DEVICE and MODEL if required.

DATASETS
Enter details of the database dataset group contents.
BLOCK and BUFFER are alternatives when specifying a PRIME dataset group.

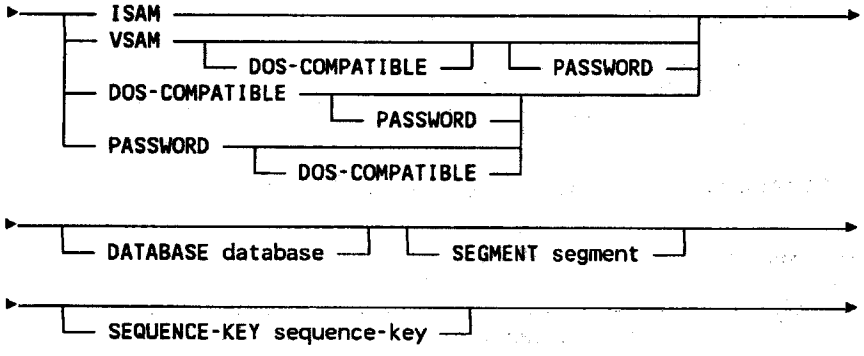
Use the ADD-TO option for continuation of a secondary dataset group segment contents list after specifying another PRIME dataset.
When ADD-TO is specified no physical blocking information applies.

Syntax



where:

index-details are:



DL1-HIDAM-DATABASE

where:

database is 1 to 8 alphanumeric characters, being the IMS name of the primary index database associated with this HIDAM database

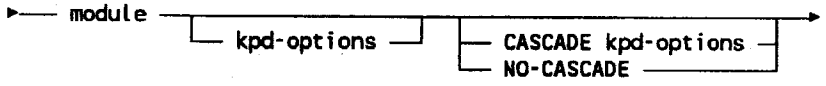
segment is 1 to 8 alphanumeric characters, being the IMS name of the primary index segment associated with this HIDAM database

sequence-key is 1 to 8 alphanumeric characters, being the IMS sequence key name of the primary index database associated with this HIDAM database.

cdcf-options are:

where:

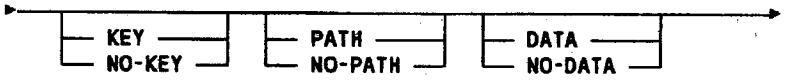
module-list is:



where:

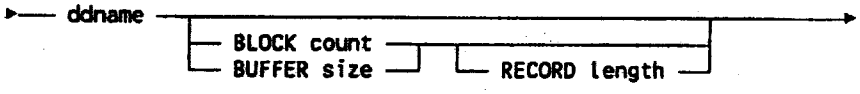
module is the name of a MODULE or PROGRAM member

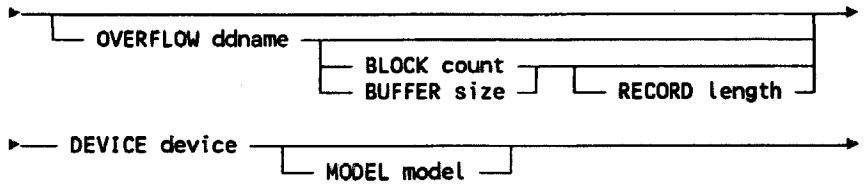
kpd-options are:



db-version is a delimited string of up to 255 characters.

i-options are:





where:

ddname is 1 to 8 alphanumeric characters, being the logical name used in the job control to identify the physical file

count, **size**, and **length** are all unsigned non-zero integers

size is an unsigned non-zero integer

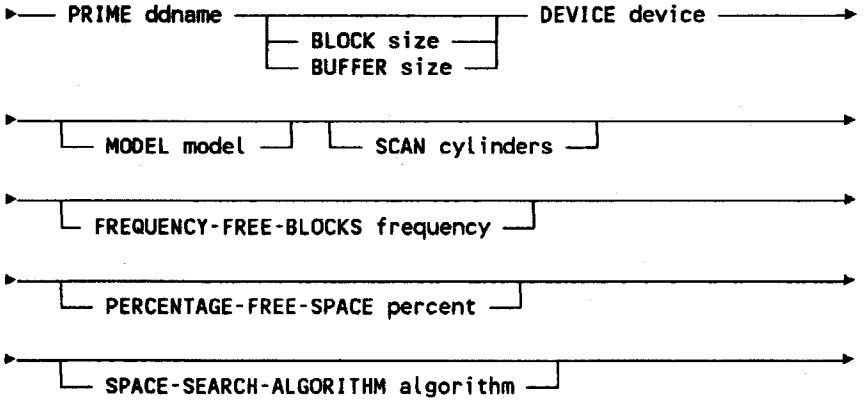
length is an unsigned non-zero integer

device is one of the keywords or numbers from the list:

```
DRUM 2311 3310 3350
CELL 2314 3330 3370
2301 2319 3340 3375
2305 2321 3344 3380
3390
```

model is an integer, 1 or 2 if device is 2305, or 1 or 11 if device is 3330

dset-details are:



where

ddname, size, device, and model are as defined above

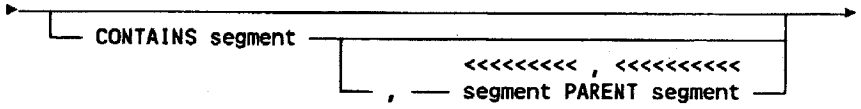
cylinders is an unsigned integer in the range 0 to 255

frequency is an unsigned integer in the range 2 to 100, or is 0

percent is an unsigned integer in the range 0 to 99

algorithm is an unsigned integer in the range 0 to 2.

contains-options are:



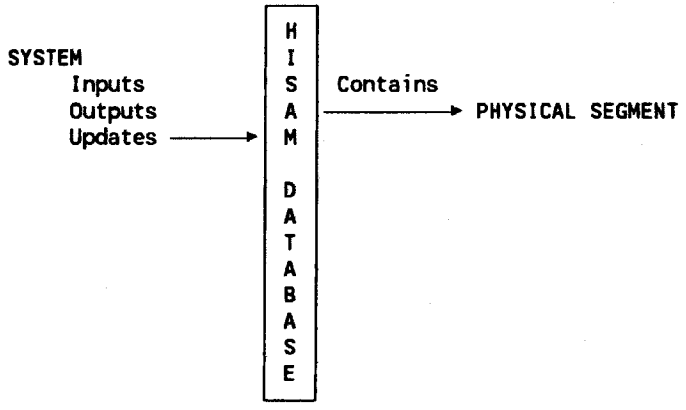
where **segment** is the name of any physical segment.

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

DL1-HISAM-DATABASE

A Hierarchical Indexed Sequential Access Method (HISAM) DATABASE allows the storage of IMS segments for predominantly read access.
The more commonly used relationships to and from HISAM PHYSICAL DATABASE are:



For further details refer to the "IMS (DL/1) Interface" documentation.

SIMPLE
Select for a Simple Hierarchical Indexed Sequential Access Method (SHISAM) database organisation. If this is the case only one segment should be included in the CONTAINS clause.
For a HISAM database organisation this does not apply.
(SIMPLE is only displayed for new members or SHISAM databases).

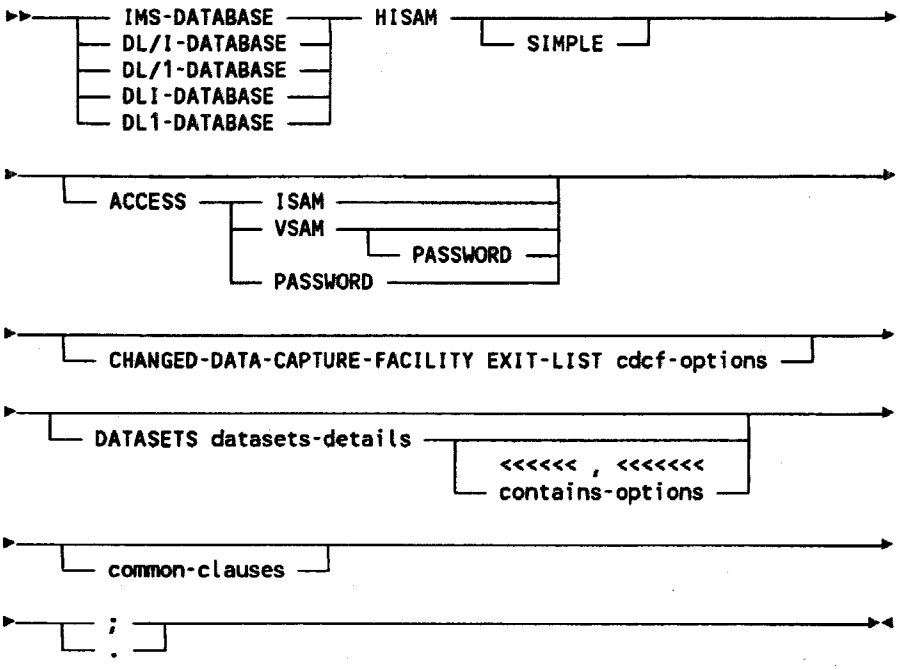
ACCESS
Defines the access method for the database dataset(s), which may be OSAM or VSAM for H/D databases,
ISAM or VSAM for HISAM databases,
and VSAM or BSAM for GSAM databases.
PASSWORD MAY ALSO BE SPECIFIED FOR PASSWORD PROTECTION.

CHANGED-DATA-CAPTURE-FACILITY
Defines the DCDCF exits with any options and the d/b version string

DL1-HISAM-DATABASE

DATASETS
Specifies the database dataset group ddname and allocation details.
BLOCK count and RECORD length are an alternative to BUFFER size.
The segments contained within the dataset(s) should also be
specified in hierarchical sequence.

Syntax

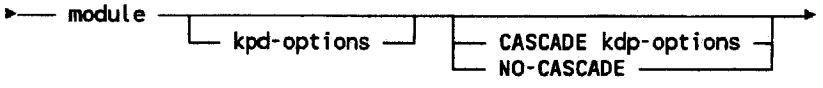


where:

cdcf-options are:

where:

module-list is:



where:

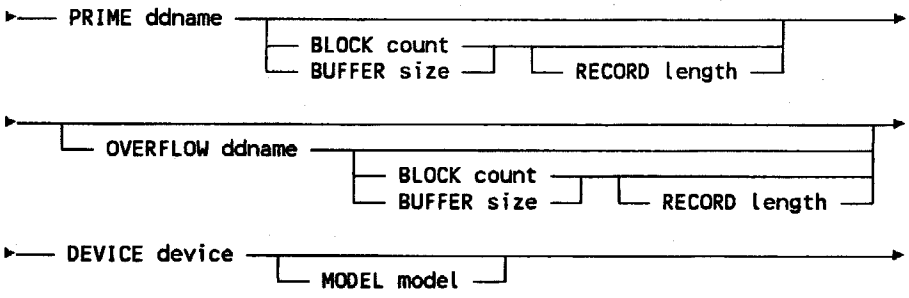
module is the name of a MODULE or PROGRAM member

DL1-HISAM-DATABASE

kpd-options are:

db-version is a delimited string of up to 255 characters

datasets-details are:



where:

ddname is 1 to 8 alphanumeric characters, being the logical name used in the job control to identify the database data set

count is an unsigned, non-zero integer, being the number of logical records per physical block

size is an unsigned, non-zero integer, being the number of bytes required per physical block or control interval.

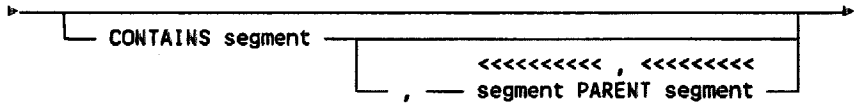
length is an unsigned non-zero integer, being the maximum length (in bytes) of a logical record. If VSAM is the operating system access method, length must be an even value.

device is one of the keywords or numbers from the list:

```
DRUM 2311 3310 3350
CELL 2314 3330 3370
2301 2319 3340 3375
2305 2321 3344 3380
3390
```

model is an integer, 1 or 2 if device is 2305, or 1 or 11 if device is 3330

contains-details are:



where:

segment is the name of a physical segment.

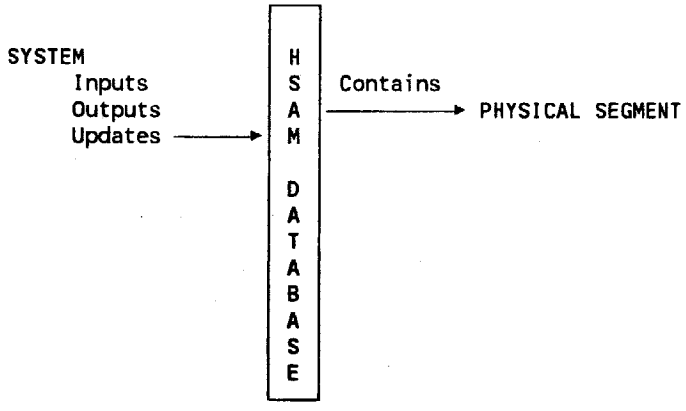
common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

DL1-HSAM-DATABASE

A Hierarchical Sequential Access Method (HSAM) DATABASE allows the storage of IMS segments for sequential read access.

The more commonly used relationships to and from HSAM PHYSICAL DATABASE are:



For further details refer to the "IMS (DL/1) Interface" documentation.

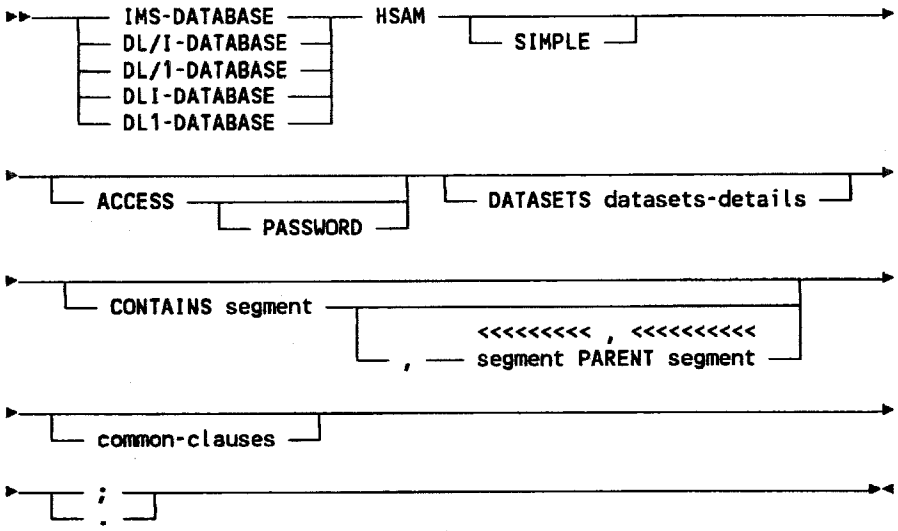
SIMPLE
Select for a Simple Hierarchical Sequential Access Method (SHSAM) database organisation. If this is the case only one segment should be included in the CONTAINS clause.
For a HSAM database organisation this attribute does not apply (SIMPLE is only displayed for new members or SHSAM databases).

ACCESS
Enter PASSWORD if required (that is the keyword "PASSWORD")

DATASETS
Input and output database dataset allocation details.

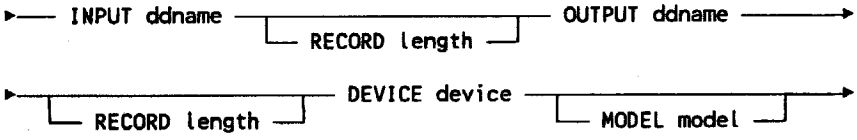
CONTAINS
Construction of segment hierarchy for HSAM databases.

Syntax



where

datasets-details is:



where:

ddname is 1 to 8 alphanumeric characters, being the logical name used in the job control to identify the physical file

length is the maximum length (in bytes) of a logical record. If VSAM is the operating system access method, length must be an even value.

device is one of the keywords or numbers from the list:

DL1-HSAM-DATABASE

```
DRUM 2311 3310 3370
CELL 2314 3330 3375
TAPE 2319 3340 3380
3390 2301 2321 3344
3400 2305 2400 3350
3420
```

model is an integer, 1 or 2 if device is 2305, or 1 or 11 if device is 3330

segment is the name of any physical segment.

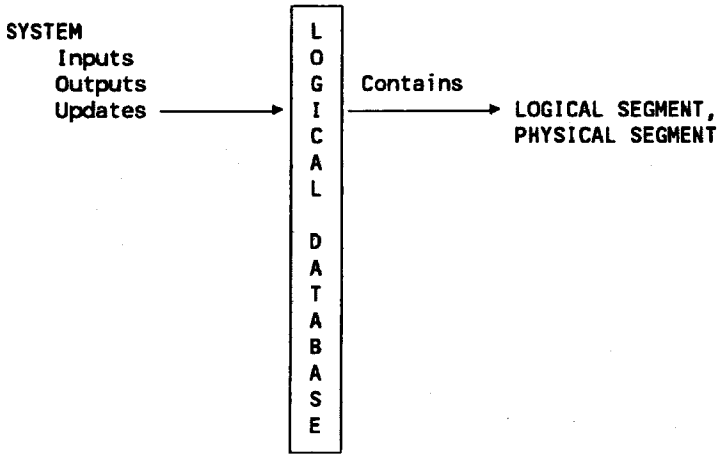
common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

DL1-LOGICAL-DATABASE

A LOGICAL DATABASE allows the definition of a logical database structure incorporating one or more physical database structures which may be joined according to the logical links implemented.

The more commonly used relationships to and from LOGICAL DATABASE are:

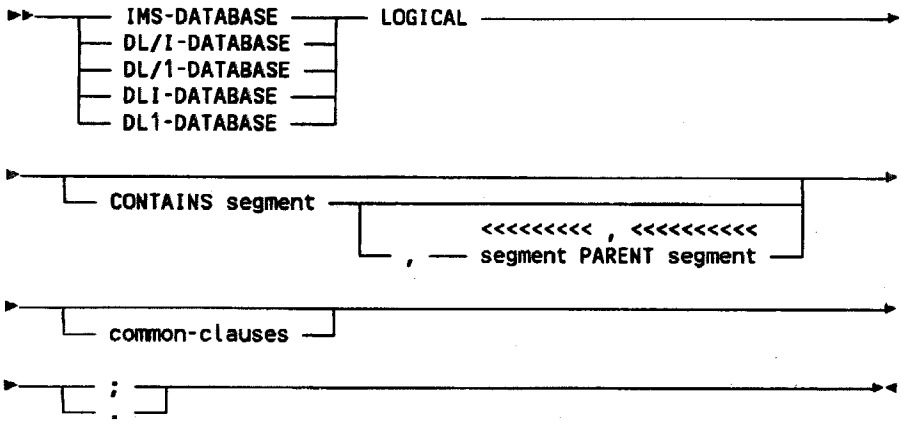


For further details refer to the "IMS (DL/1) Interface" documentation.

CONTAINS
Construction of segment hierarchy for logical database
(physical segments, prefix SP-, may also be contained directly).

DL1-LOGICAL-DATABASE

Syntax



where

segment is the name of a logical or physical segment

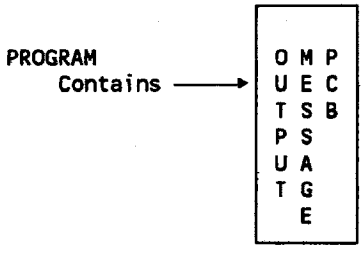
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

DL1-OUTPUT-MESSAGE-PCB

An OUTPUT MESSAGE PCB defines a message processing program communication block (PCB).

The more commonly used relationships to and from OUTPUT MESSAGE PCB are:



For further details refer to the "IMS (DL/1) Interface" documentation.

NAME
Name another PCB upon which the attributes of this PCB will be based. If specified then only common clauses should follow.

LOGICAL-TERMINAL
No help text defined

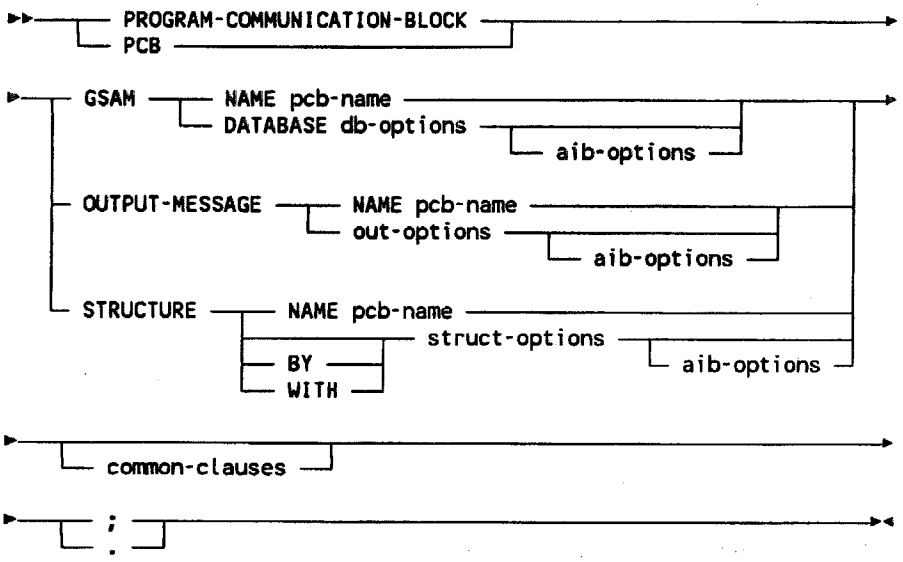
TRANSACTION-CODE
No help text defined

MODIFIABLE
No help text defined

AIB-LIST-ADDRESS
Enter YES or NO to signify whether this PCB address is required in the application interface-control block for IMS releases from 2.0. Will cause generation of LIST = YES/NO when the PRODUCE IMS command should also specify OPTIONS including PCB-NAME.

DL1-OUTPUT-MESSAGE-PCB

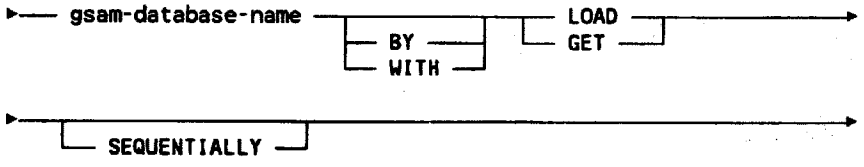
Syntax



where

pcb-name is the name of another PROGRAM-COMMUNICATION-BLOCK member

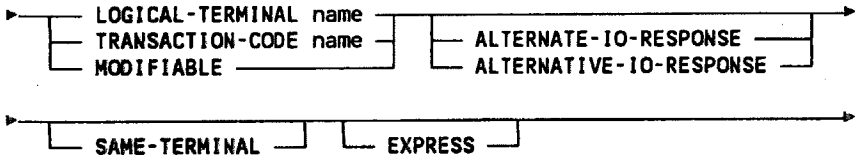
db-options are:



where **gsam-database-name** is the name of a database member of the GSAM type.

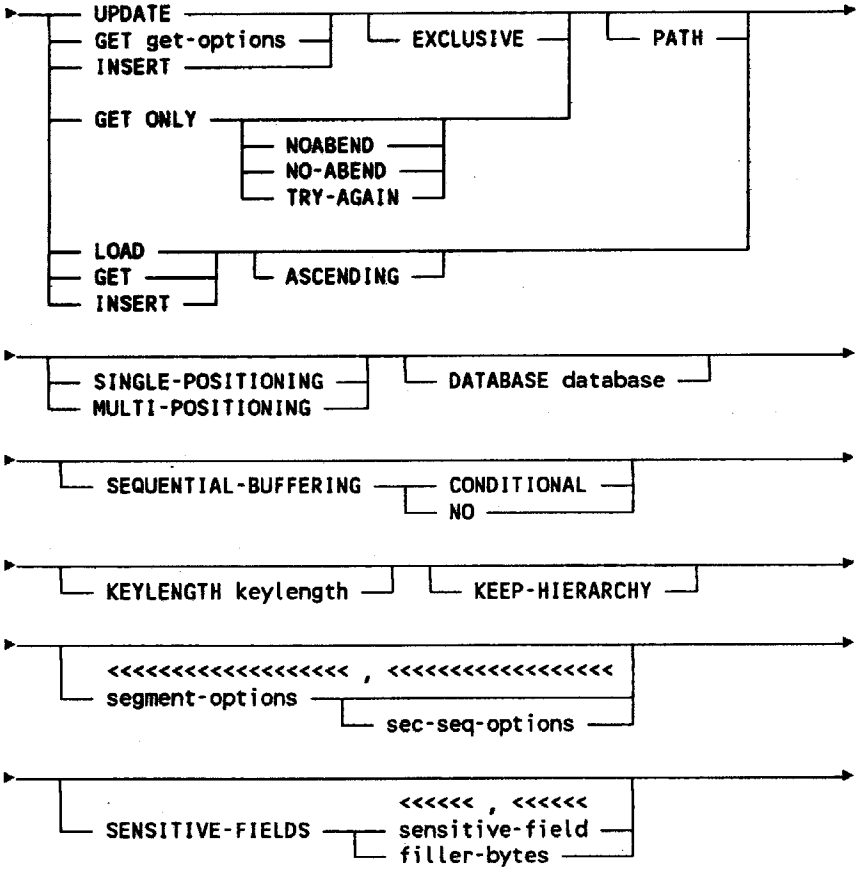
aib-options are:

out-options are:



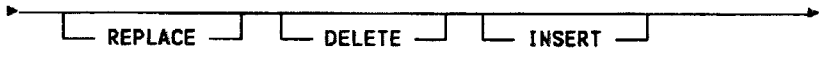
where **name** is an alphanumeric name 1 to 8 characters in length.

structure-options are:



where

get-options are:

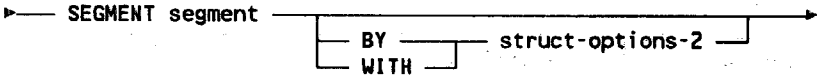


Note: The UPDATE, GET, INSERT, LOAD, EXCLUSIVE, ASCENDING, PATH, GET ONLY, NOABEND, TRY-AGAIN, REPLACE and DELETE keywords can all be optionally separated by commas.

database is the name of a DATABASE member

keylength is an integer in the range 0 to 32767

segment-options are:

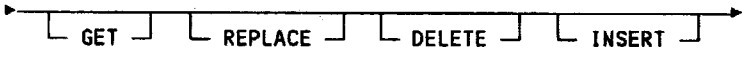


where

segment is the name of a SEGMENT member

struct-options-2 are:

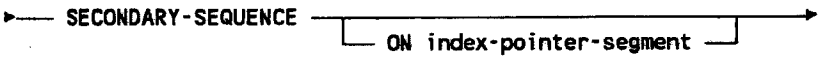
where **grdi-options** are:



Note 1: The UPDATE, KEY-SENSITIVE, GET, REPLACE, DELETE, INSERT, EXCLUSIVE and PATH keywords can all be optionally separated by commas.

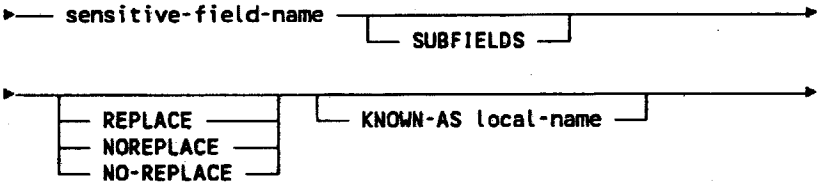
Note 2: You must specify at least one keyword in **grdi-options**.

sec-seq-options are:



where **index-pointer-segment** is the name of an INDEX-POINTER SEGMENT member.

sensitive-field is:



where:

local-name is a name, conforming to the rules for member names stated in the CONTROLMANAGER User's Guide

sensitive-field-name is the name of a GROUP, ITEM or sequence key member or concatenated key member

filler-bytes is an unsigned integer in the range 1 to 32767

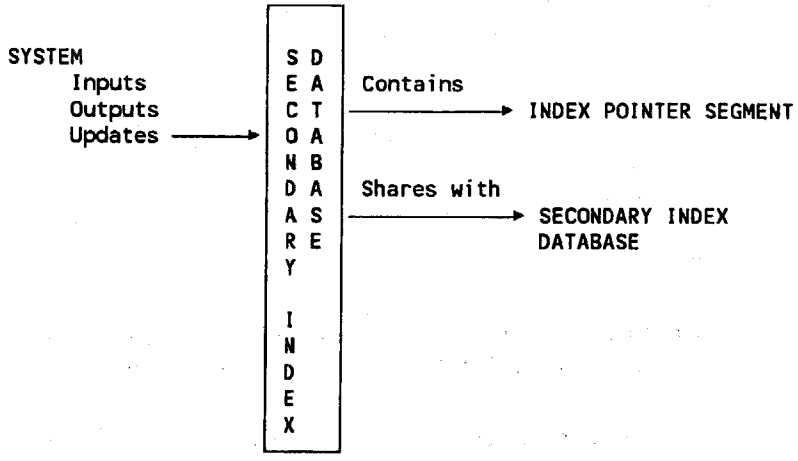
common-clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DL1-SECONDARY-INDEX-DATABASE

A SECONDARY INDEX DATABASE records the dataset information for a secondary index.

The more commonly used relationships to and from SECONDARY INDEX DATABASE are:



For further details refer to the "IMS (DL/1) Interface" documentation.

SHARES-WITH
Specifies the name of another secondary index database which is shared with this index database. If this is required then secondary index ACCESS and DATASETS clauses should be deleted.

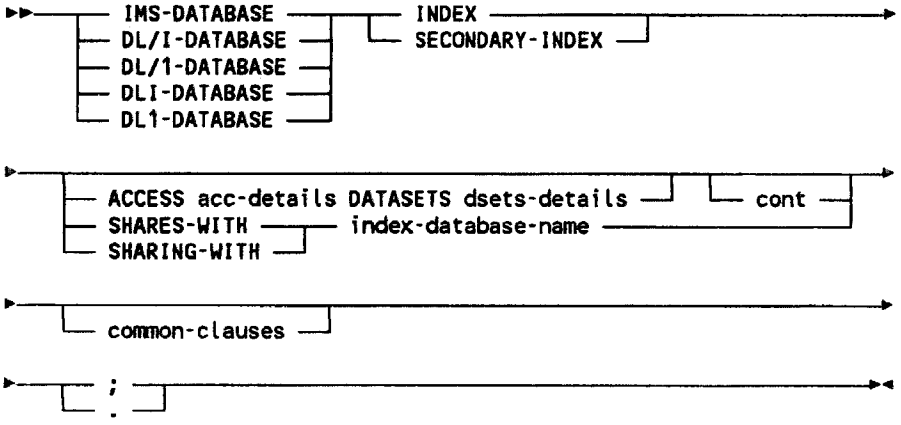
ACCESS
Defines the access method for the database dataset(s), which may include VSAM, DOS-COMPATIBLE, or PASSWORD and may be PROTECTED or NOT-PROTECTED.

DATASETS
Defines the secondary index database dataset allocation details. BLOCK count and RECORD length are an alternative to BUFFER size.

CONTAINS
The CONTAINS clause specifies the index pointer segment for this secondary index.

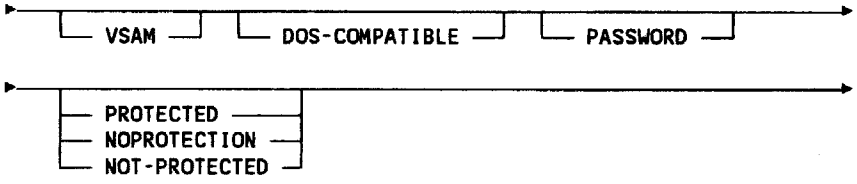
DL1-SECONDARY-INDEX-DATABASE

Syntax



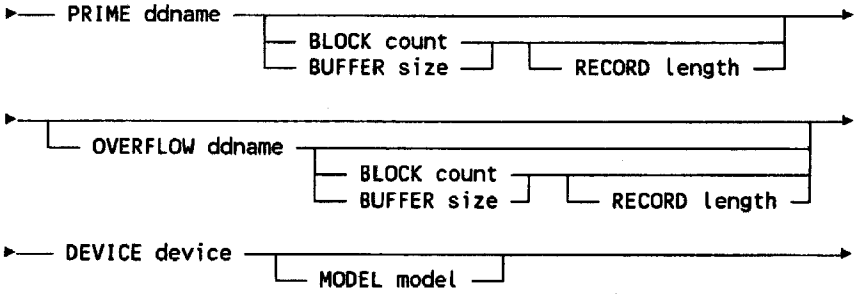
where:

acc-details are:



index-database-name is the name of another IMS(DL/I) SECONDARY-INDEX database

dsets-details are:



where:

ddname is 1 to 8 alphanumeric characters, being the logical name used in the job control to identify the physical file

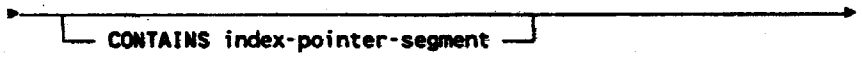
count, size and **length** are all unsigned non-zero integers

device is one of the keywords or numbers from the list:

```
DRUM 2311 3310 3350
CELL 2314 3330 3370
2301 2319 3340 3375
2305 2321 3344 3380
3390
```

model is an integer, 1 or 2 if device is 2305, or 1 or 11 if device is 3330

cont is:



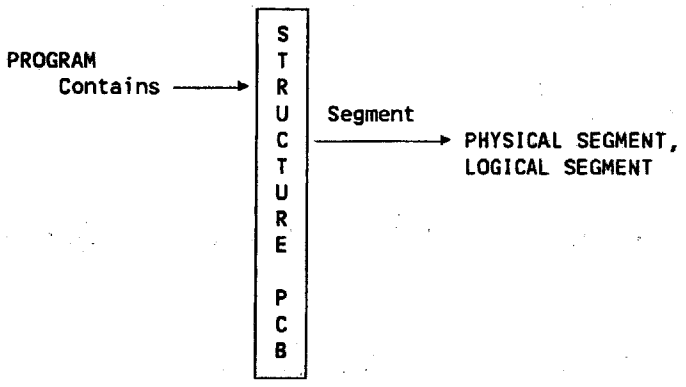
where **index-pointer-segment** is an INDEX-POINTER-SEGMENT member

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

DL1-STRUCTURE-PCB

A STRUCTURE PCB defines the segment structure and processing options for a database processing program communication block (PCB). The more commonly used relationships to and from STRUCTURE PCB are:



For further details refer to the "IMS (DL/1) Interface" documentation.

NAME
Name another PCB upon which the attributes of this PCB will be based. If specified then only common clauses should follow.

BY
Specify database PCB processing options (with spaces inbetween) to be applied to any segments with no processing options specified. Valid procopts are; GET REPLACE INSERT DELETE PATH
UPDATE EXCLUSIVE
LOAD ASCENDING
ONLY NO-ABEND TRY-AGAIN
combinations of these may vary in validity.

DATABASE
Name of the database which this PCB references.

SINGLE-POSITIONING
Select if single positioning is required for this database PCB. This is an alternative to multi-positioning processing.

MULTI-POSITIONING
Select if multiple positioning is required for this database PCB.
This is an alternative to single positioning processing.

SEQUENTIAL-BUFFERING
Specifies whether sequential buffering is to be used for ISAM/OSAM process optimisation.

Only applies from IMS/VS version 2.0 onwards.

KEYLENGTH
MANAGER Products will calculate this value when generating a PSB definition, but this may require a performance overhead.
Specifying the maximum key length here will override the need for MANAGER Products to calculate the value.

KEEP-HIERARCHY
Select if it is desired to keep the specified segment hierarchy structure, without allowing MANAGER Products to re-order segments or to insert missing segments.

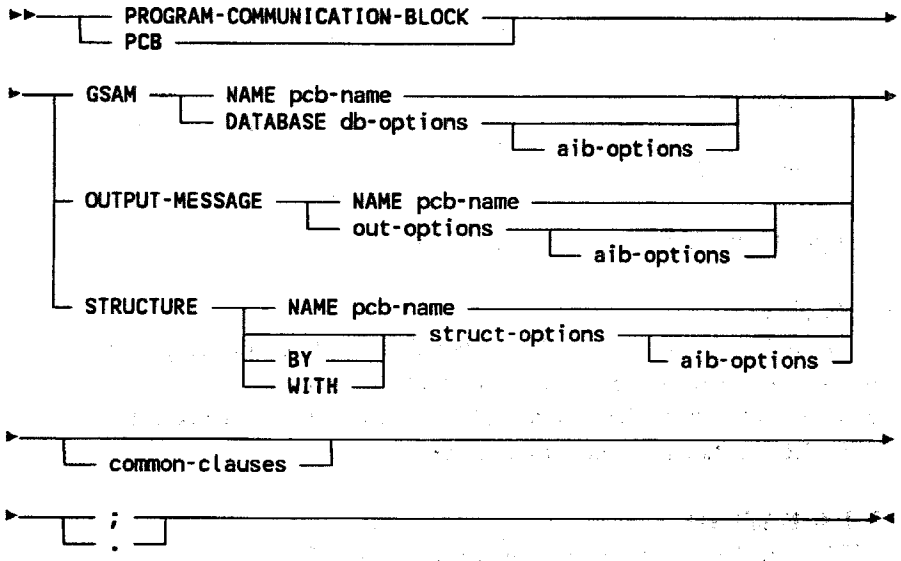
If the hierarchy is invalid this will still be detected when a PSB definition is generated.

SEGMENT
Name segments to which this PCB should be sensitive and specify required processing options (with spaces inbetween) to be applied specifically to this segment.
Valid procopts are; GET REPLACE INSERT DELETE PATH
UPDATE EXCLUSIVE
KEY-SENSITIVE
combinations of these may vary in validity.
A secondary processing sequence may be specified by naming an index pointer segment, if applicable.

AIB-LIST-ADDRESS
Enter YES or NO to signify whether this PCB address is required in the application interface-control block for IMS releases from 2.0.
Will cause generation of LIST = YES/NO when the PRODUCE IMS command should also specify OPTIONS including PCB-NAME.

DL1-STRUCTURE-PCB

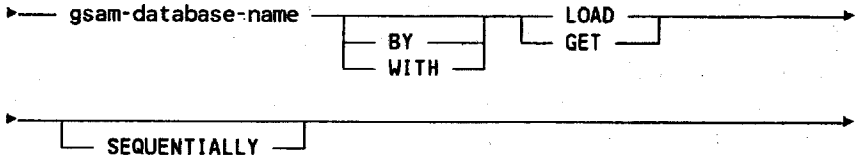
Syntax



where

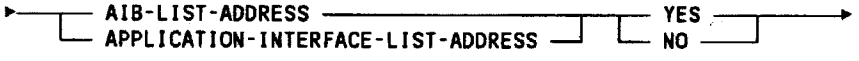
pcb-name is the name of another PROGRAM-COMMUNICATION-BLOCK member

db-options are:

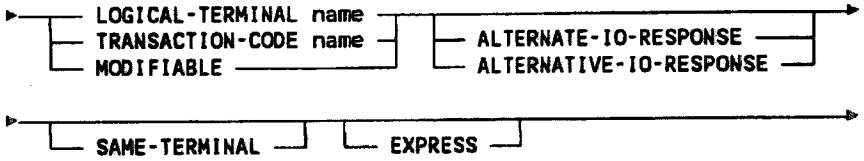


where **gsam-database-name** is the name of a database member of the GSAM type.

aib-options are:

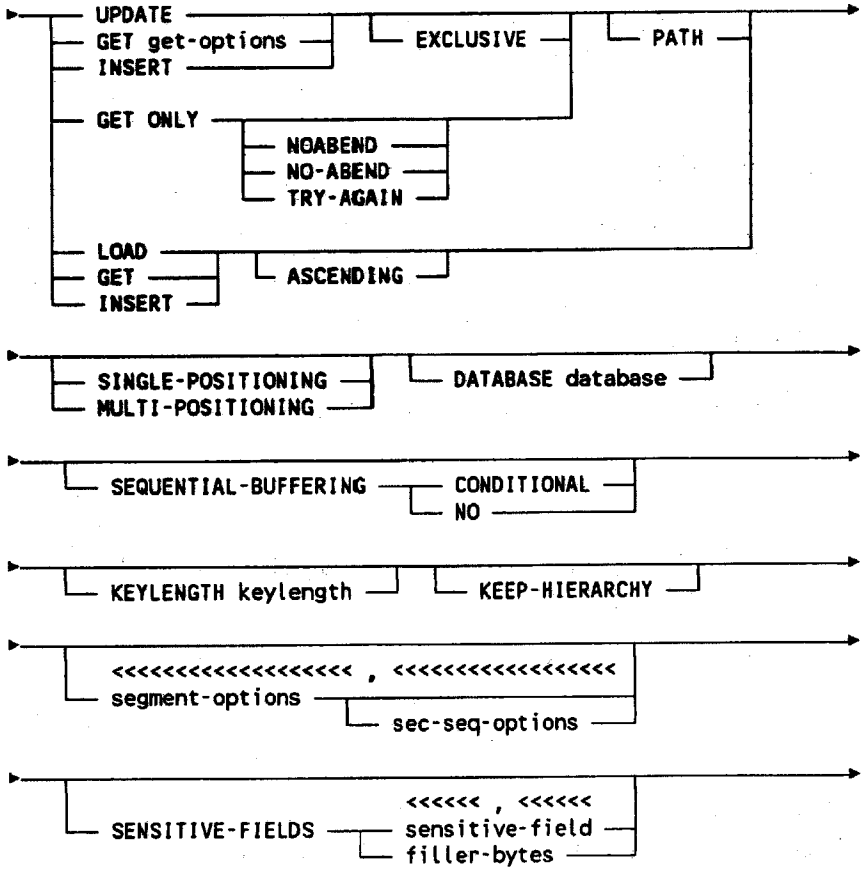


out-options are:



where **name** is an alphanumeric name 1 to 8 characters in length.

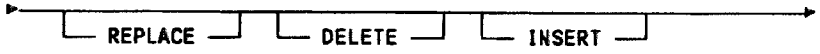
structure-options are:



DL1-STRUCTURE-PCB

where

get-options are:

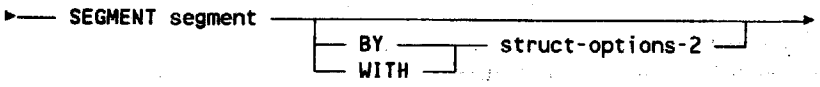


Note: The UPDATE, GET, INSERT, LOAD, EXCLUSIVE, ASCENDING, PATH, GET ONLY, NOABEND, TRY-AGAIN, REPLACE and DELETE keywords can all be optionally separated by commas.

database is the name of a DATABASE member

keylength is an integer in the range 0 to 32767

segment-options are:

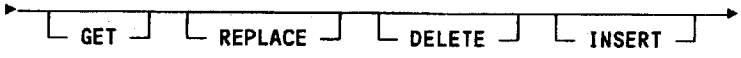


where

segment is the name of a SEGMENT member

struct-options-2 are:

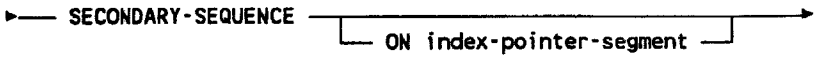
where **grdi-options** are:



Note 1: The UPDATE, KEY-SENSITIVE, GET, REPLACE, DELETE, INSERT, EXCLUSIVE and PATH keywords can all be optionally separated by commas.

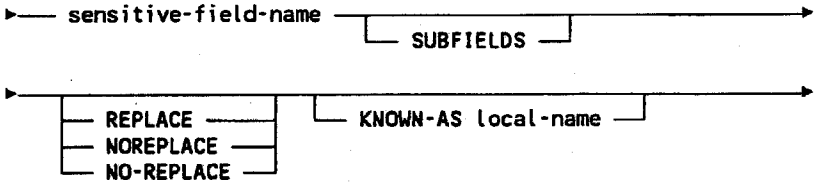
Note 2: You must specify at least one keyword in **grdi-options**.

sec-seq-options are:



where **index-pointer-segment** is the name of an INDEX-POINTER SEGMENT member.

sensitive-field is:



where:

local-name is a name, conforming to the rules for member names stated in the CONTROLMANAGER User's Guide

sensitive-field-name is the name of a GROUP, ITEM or sequence key member or concatenated key member

filler-bytes is an unsigned integer in the range 1 to 32767

common-clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

DOCUMENT

By defining the DOCUMENT member type you have the ability to specify a repository document. A repository document includes entries which consist of text and/or commands. The repository document becomes an output document which includes the result of commands, references to subdocuments and text.

TITLE
Contains a text string of up to 32767 delimited character strings, each up to 252 characters long. It should contain any information which you wish to record, but which isn't administrative data.

CONTENTS
Specify the contents of documentation. The following specifications are possible:

- free form text
- tool commands
- relationships to subdocuments, which are expanded at run time.

Syntax

The syntax of the DOCUMENT member type will be supplied in a future issue.

ENTITY

You use an ENTITY member to define a business entity.

For details of business entities refer to the METHODMANAGER Administration manual (MMR-ADMIN).

IDENTIFIER IS
To specify the identifying attribute, enter:

```
IDENTIFIER IS name
```

name is the name of an ITEM member or a GROUP member.

ONE-ATTRIBUTES ARE
May contain the name of one or more data elements.
Elements named in this clause occur only once in the member.

ONE-ASSOCIATIONS
Lists the entities with which this entity has a one-association. The one-association can be given a label indicating the action of this member on the members listed in the clause.
This is done by completing the LABEL subclause.
eg) ONE-ASSOCIATION
LABEL "ORDERED-BY"
TO EN-CUSTOMER
Additional information about the associations of this member can be recorded in the ASSOCIATION-DETAILS clause.

MULTI-ASSOCIATIONS
This clause may contain the name of an entity (after the "TO").
A multi-association occurs when an entity has an association with more than one occurrence of another entity. This clause names the other entity with which this entity has the multi-association.
The multi-association can be labelled to indicate the action of this entity on the entity named in the clause. This is done by entering the name of the association in the LABEL subclause.
The cardinality of the relationship can be recorded before the LABEL subclause, for example:

```
MULTI-ASSOCIATION
100
LABEL "ORDERS"
TO  EN-CUSTOMER-ORDER
```

Additional information about the associations of this member can be recorded in the ASSOCIATION-DETAILS clause.

SUB-ENTITIES ARE
Lists the sub-entities which make up this member. Every member which is a sub-entity of this one must be named in this clause. No member must be named which is not a sub-entity of this member.

MAXIMUM-OCCURRENCE
To specify the maximum number of occurrences of the entity, enter:

```
MAXIMUM-OCCURRENCE integer
```

integer is a positive integer.

MINIMUM-OCCURRENCE
To specify the minimum number of occurrences of the entity, enter:

```
MINIMUM-OCCURRENCE integer
```

OCCURRENCE
To specify the expected number of occurrences of the entity, enter:

```
OCCURRENCE integer
```

integer is a positive integer within the OCCURRENCE range.

ASSOCIATION-DETAILS
Contains a full description of each association named in the ONE-ASSOCIATION TO and MULTI-ASSOCIATION TO clauses. The names of these associations may be recorded in their LABEL subclauses.

GROWTH-RATE-PERIOD
To specify the growth rate period, enter:

```
GROWTH-RATE-PERIOD 'string'
```

string is one of the following:

```
MONTH
DAY
YEAR
QUARTER
etc.
```

To specify the growth rate percentage, enter:

```
GROWTH-RATE-PERCENT integer
```

integer is any integer.

All these clauses are optional and have no default values.

ADEQUATELY-NORMALIZED
To specify that the entity is adequately normalized, enter:

```
ADEQUATELY-NORMALIZED Y
```

To specify that the entity is not adequately normalized, enter:

```
ADEQUATELY-NORMALIZED N
```

NORMALIZATION-LEVEL
To specify the level of normalization, enter:

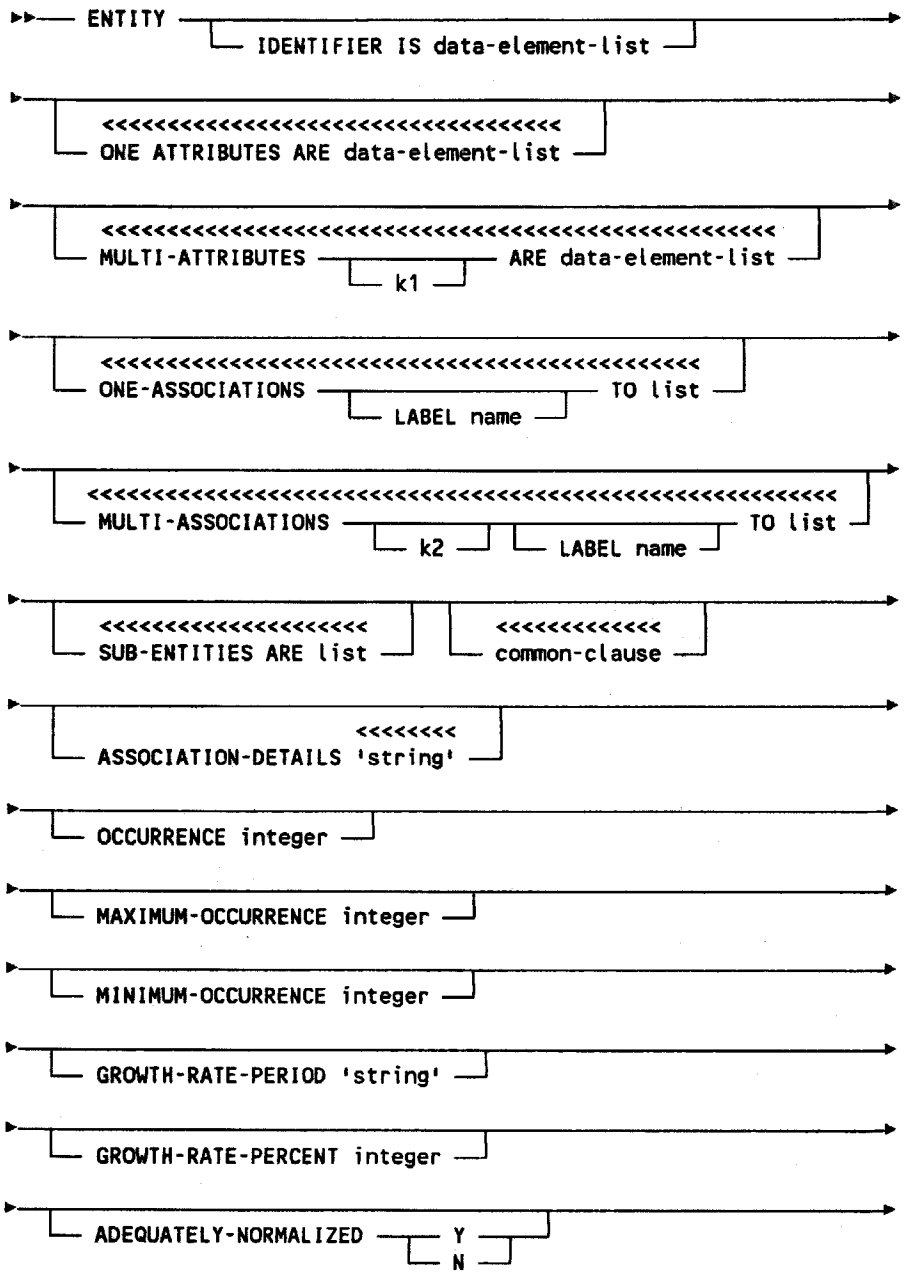
```
NORMALIZATION-LEVEL level
```

level is one of the following:

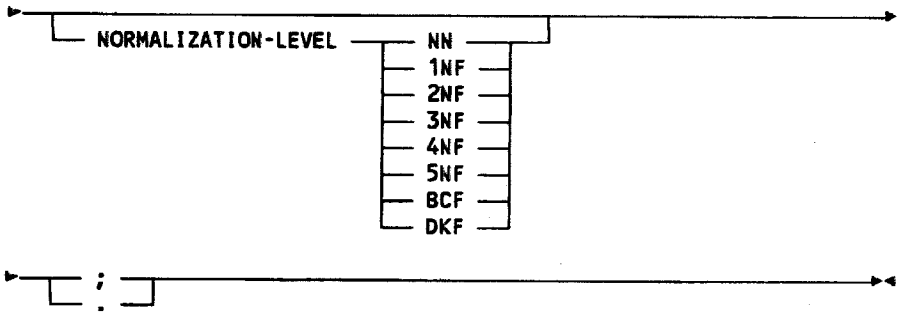
```
NN      (not normalized)
1NF     (first normal form)
2NF     (second normal form)
3NF     (third normal form)
4NF     (fourth normal form)
5NF     (fifth normal form)
BCF     (Boyce-Codd normal form)
DKF     (domain key normal form).
```

All these clauses are optional and have no default values.

Syntax


ENTITY
IDENTIFIER IS data-element-list
ONE ATTRIBUTES ARE data-element-list
MULTI-ATTRIBUTES
k1
ARE data-element-list
ONE-ASSOCIATIONS
LABEL name
TO list
MULTI-ASSOCIATIONS
k2
LABEL name
TO list
SUB-ENTITIES ARE list
common-clause
ASSOCIATION-DETAILS 'string'
OCCURRENCE integer
MAXIMUM-OCCURRENCE integer
MINIMUM-OCCURRENCE integer
GROWTH-RATE-PERIOD 'string'
GROWTH-RATE-PERCENT integer
ADEQUATELY-NORMALIZED
Y
N

ENTITY



where

data-element-list is:

where

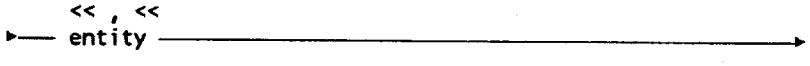
item is the name of an item

group is the name of a group.

k1 is an unsigned integer indicating the multiplicity of the specified set of one or more multi-attributes, that is, the average number of values of the listed data element (or set of data elements) determined by a single value of the identifier of the given entity

name is a string of up to 32 characters

list is:



where **entity** is the name of an entity.

k2 is an unsigned integer indicating the multiplicity of the specified multi-association, that is, the expected number of instances of the listed target entity (or set of target entities) determined by a single instance of the given entity.

common-clause is a common clause.

Refer to Appendix 2 for details of the common clauses.

string is a string of up to 32 characters

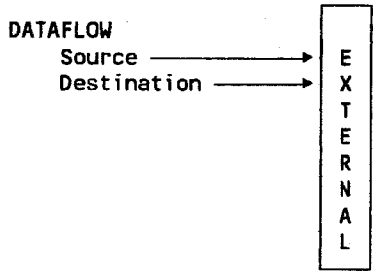
integer is an integer.

Note: the delimiters shown in the above syntax are required when defining an ENTITY member via the command interface.

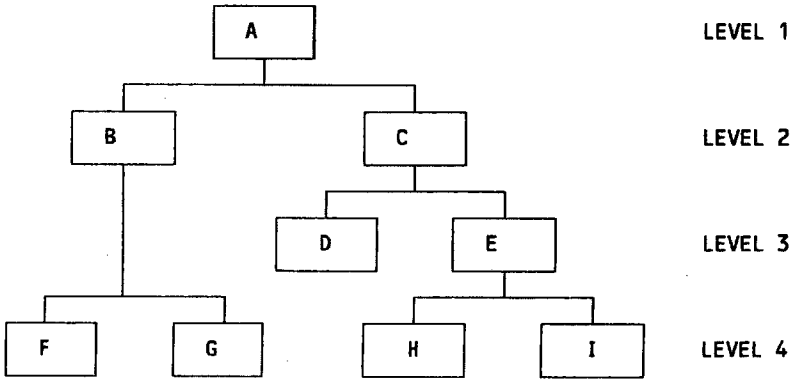
EXTERNAL

An EXTERNAL is an entity or process which lies outside the area under study and is the source or destination of one or more dataflows.

The more commonly used relationships to and from EXTERNAL are:



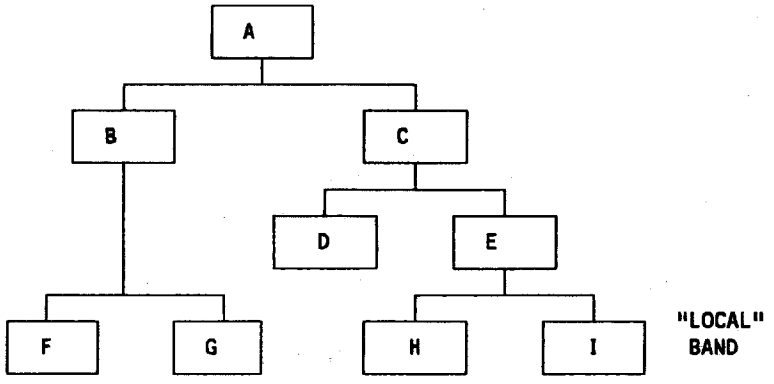
LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy. The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

BAND

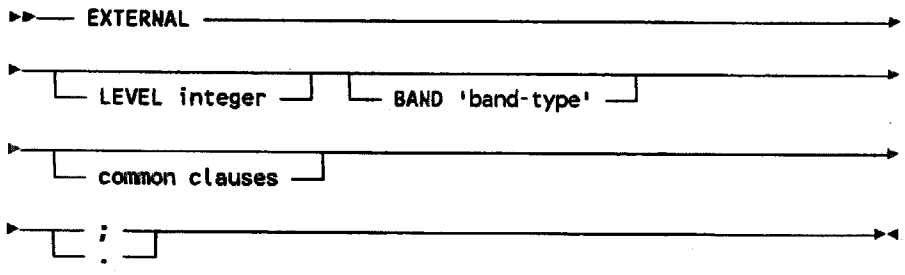
The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.



The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

EXTERNAL

Syntax



where

integer is an integer value of up to 18 digits, optionally preceded by a sign

'band-type' is a text string of up to 78 characters, including delimiters.

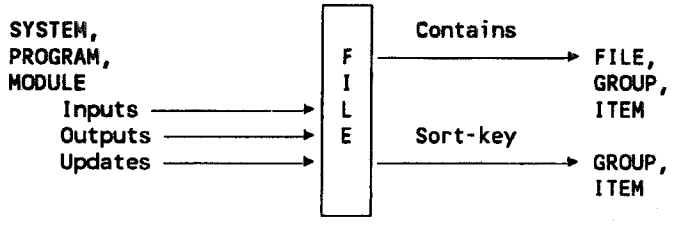
Note: the delimiters shown in the above syntax are required when defining an EXTERNAL member via the command interface.

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

FILE

A FILE is a set of related records treated as a unit.
The more commonly used relationships to and from FILE are:



HELD-AS
This clause is used if the member contains items and groups in the HELD-AS form. It is an alternative to ENTERED-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

ENTERED-AS
This clause is used if the member contains items and groups in the ENTERED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

REPORTED-AS
This clause is used if the member contains items and groups in the REPORTED-AS form.It is an alternative to HELD-AS, DEFAULTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

DEFAULTED-AS
This clause is used if the member contains items and groups in the DEFAULTED-AS form. It is an alternative to HELD-AS, REPORTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

ALIGNED
This clause is used if all the members in the CONTAINS clause are ALIGNED. It is an alternative to UNALIGNED or NOT-ALIGNED. Only one such clause is valid in the definition.

FILE

UNALIGNED
This clause is used if all the members in the CONTAINS clause are UNALIGNED. It is an alternative to ALIGNED.
Only one such clause is valid in the definition.

NOT-ALIGNED
This clause is used if all the members in the CONTAINS clause are NOT-ALIGNED. It is an alternative to ALIGNED.
Only one such clause is valid in the definition.

CONTAINS
Contains the names of one or more FILE, GROUP and ITEM members, which compose the structure of this file.
Examples:
To express the fact that the file contains one type of record:

```
GR-RECORD
```

To express the fact that the file also contains a header and trailer and a record-type flag in each record:

```
IT-RECORD-TYPE
GR-HEADER IF IT-RECORD-TYPE = 1
ELSE GR-RECORD IF IT-RECORD-TYPE = 2
ELSE GR-TRAILER IF IT-RECORD-TYPE = 9
```

SORT-KEY
Specifies the component members on which the contents of this member are sorted. For each member named in this clause, the sort sequence (ASCENDING or DESCENDING) must be specified.

DIRECT
This clause is used if the file format is DIRECT.

INDEXED
This clause is used if the file format is INDEXED.

KEYED
This clause is used if the file format is KEYED.

PARTITIONED
This clause is used if the file format is PARTITIONED.

SEQUENTIAL
This clause is used if the file format is SEQUENTIAL.

VSAM
This clause is used if the file format is VSAM.

FIXED
Declares the block size and record size of data in the file. The FIXED keyword indicates that the block size and record size is used throughout the whole file.

VARIABLE
Variable Length

DEVICE
Contains the name, number or model of the type of peripheral device on which this file is normally processed.
Examples:

```
DEVICE DISK 3380
DEVICE TAPE
```

DENSITY
Contains the packing density of the file in bytes per inch, or other predefined units.

VOLUME
Contains the name or serial number of the disk or tape volume, or any other medium, which contains this file.

SIZE
Records the size of the file in appropriate units. If 2 sizes are entered these represent the primary and secondary areas respectively.
Examples:

```
30 TRACKS, 3 TRACKS
15 CYLINDERS
```

STANDARD-LABELS
The file has STANDARD-LABLES. This is an alternative to NO-LABELS or USER-LABELS. Only one such clause is valid in the definition.

FILE

USER-LABELS
Contains the name of a module by which the USER-LABELS are processed. This is an alternative to STANDARD-LABELS and NO-LABELS. Only one such clause is valid in the definition.

NO-LABELS
The file has NO-LABLES. This is an alternative to STANDARD-LABELS or USER-LABELS. Only one such clause is valid in the definition.

GENERATION-CYCLE
The number of generations of this file which must be retained. Depending on updating methods, this value may be the number of previous generations, or the number of backup copies plus the current generation.

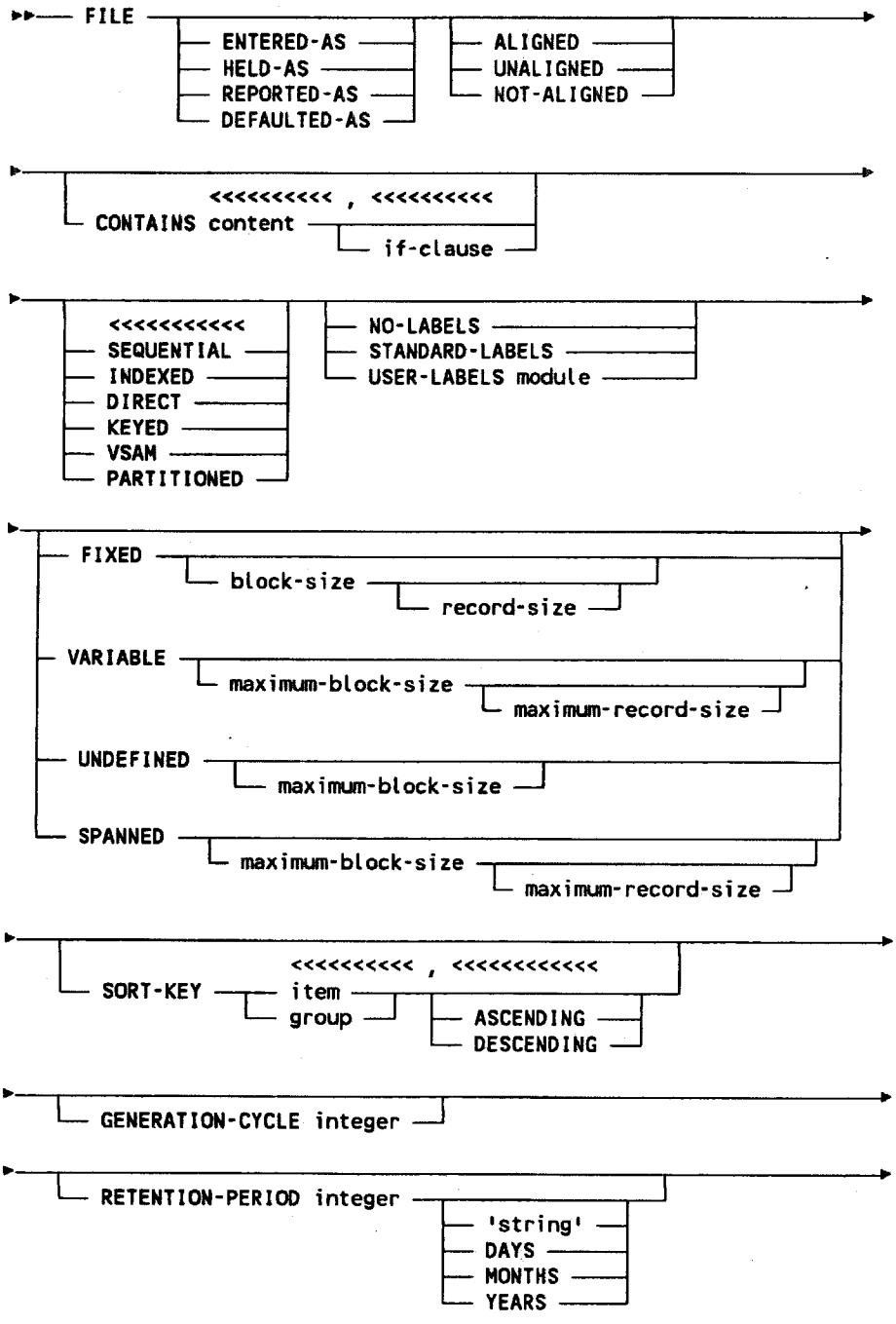
RETENTION-PERIOD
Defines the time for which a backup copy is retained. Depending on update method, this may be the period for which a previous generation is retained, or the period for which backup copies are retained. DAYS, MONTHS and YEARS are optional keywords.
Examples:

```
5 MONTHS
2 YEARS
```

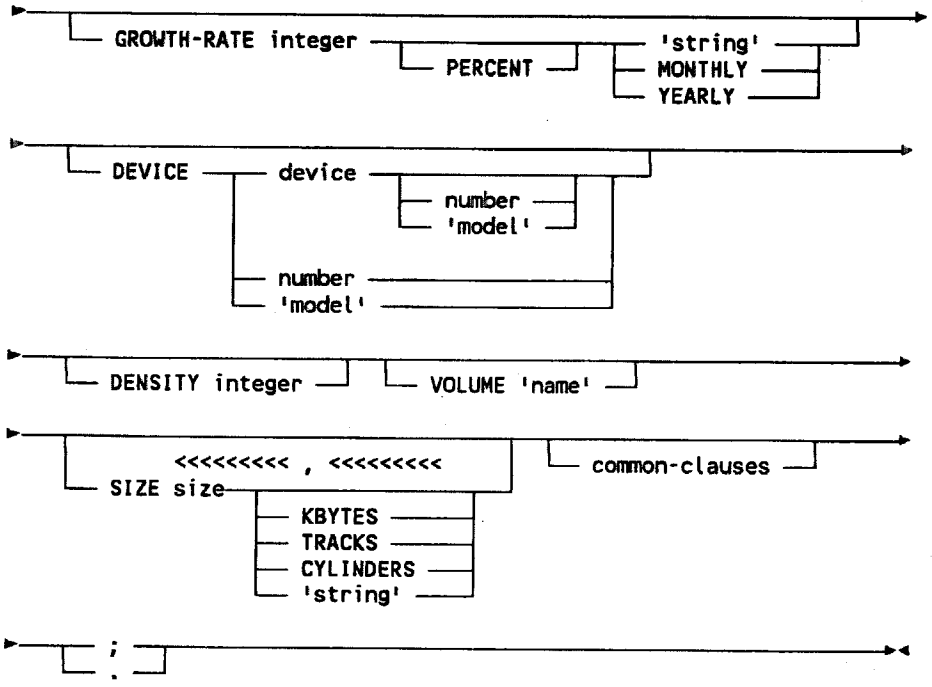
GROWTH-RATE
Average growth rate of this file over a period of time, in terms of number of records over a defined period or, if PERCENT is specified, as a percentage increase in file size over a defined period.
Examples:

```
25 PERCENT YEARLY
1  PERCENT MONTHLY
```

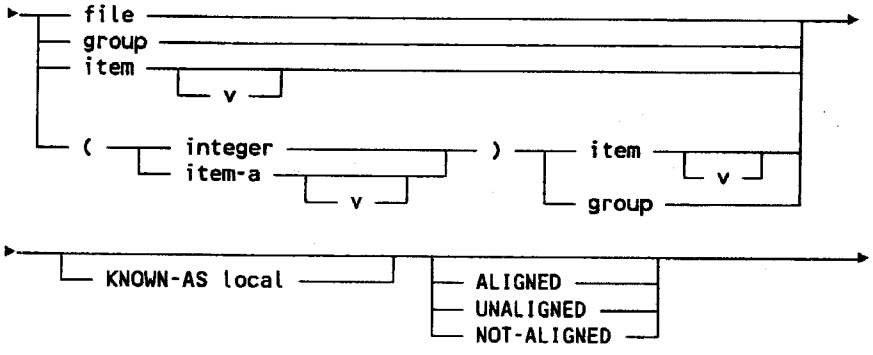
Syntax


FILE
ENTERED-AS
HELD-AS
REPORTED-AS
DEFAULTED-AS
ALIGNED
UNALIGNED
NOT-ALIGNED
CONTAINS content
<<<<<<<<<< , <<<<<<<<<<
if-clause
<<<<<<<<<<
SEQUENTIAL
INDEXED
DIRECT
KEYED
VSAM
PARTITIONED
NO-LABELS
STANDARD-LABELS
USER-LABELS module
FIXED
block-size
record-size
VARIABLE
maximum-block-size
maximum-record-size
UNDEFINED
maximum-block-size
SPANNED
maximum-block-size
maximum-record-size
SORT-KEY
<<<<<<<<<< , <<<<<<<<<<<
item
group
ASCENDING
DESCENDING
GENERATION-CYCLE integer
RETENTION-PERIOD integer
'string'
DAYS
MONTHS
YEARS

FILE



content is:



where

file is the name of a FILE member

group is the name of a GROUP member

item is the name of an ITEM member

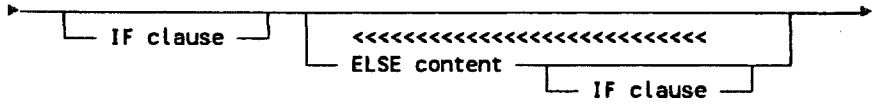
v is an unsigned integer in the range 1 to 15

integer is an unsigned integer not greater than 32767

item-a is the name of an ITEM

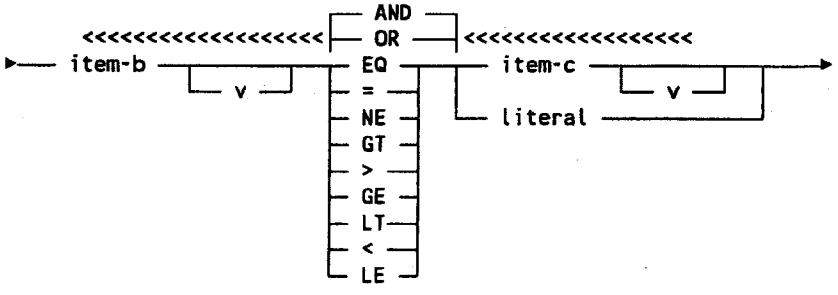
local is a name conforming to the rules for member names

if-clause is:



where

clause is:



where

item-b is the name of an ITEM to be compared with item-c or a literal

v is as defined above

item-c is the name of the ITEM to be compared with item-b

literal is the literal being compared with item-b

EQ or **=** means equal to

NE means not equal to

GT or **>** means not greater than

GE means greater than or equal to

LT or **<** means less than

LE means less than or equal to

content is as defined above

module is the name of a MODULE

block-size is an unsigned integer, being the number of characters per block

record-size is an unsigned integer, being the number of characters per record

maximum-block-size is an unsigned integer not greater than 32767

maximum-record-size is an unsigned integer not greater than 32767

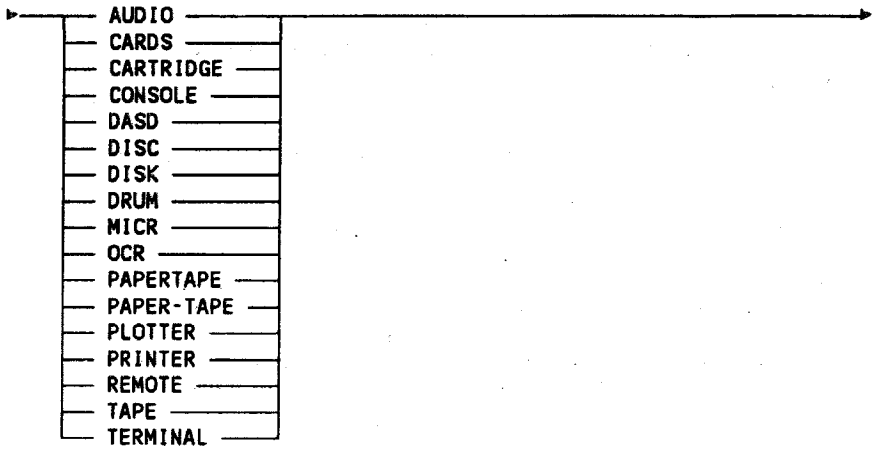
item is as defined above

group is as defined above

integer is as defined above

'string' is a character string of not more than 256 characters

device is:



number is a manufacturer's type number or model number, with a maximum of eight digits.

'**model**' is a character string of not more than eight characters, being a manufacturer's model identifier. The string may include blank spaces.

'**name**' is a character string of not more than 256 characters.

size is an unsigned integer of between 1 and 10 digits, with a maximum value of 2147483647

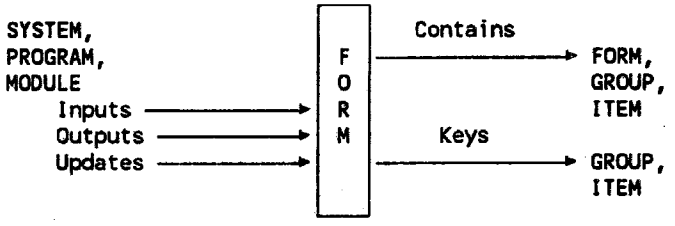
common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

FORM

A form is a formatted screen or document used for the input of information into a computer system.

The more commonly used relationships to and from FORM are:



CONTAINS
This clause may contain the names of one or more forms or data elements.

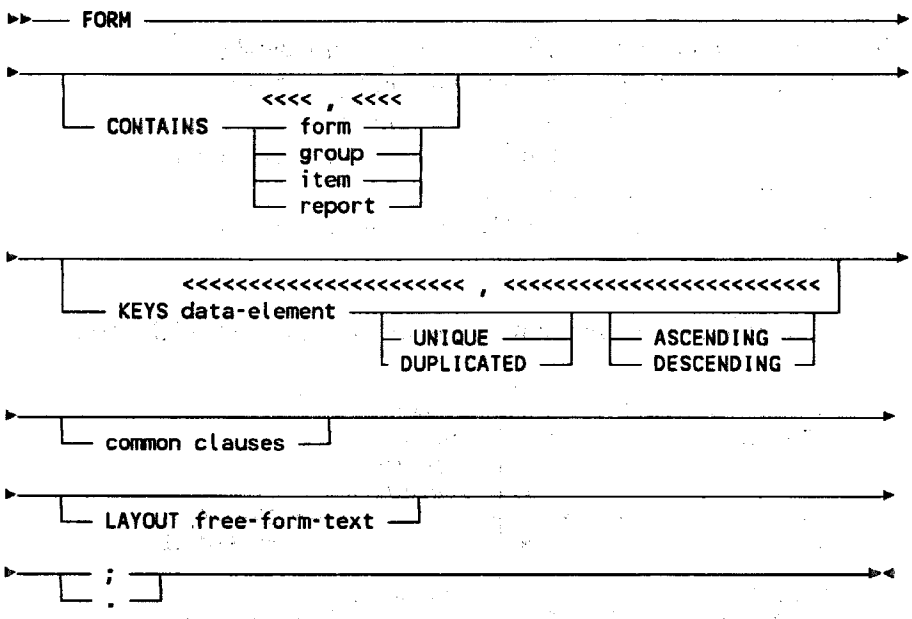
It must contain the name of every member which is a part of this form.

KEYS
Key fields of the form.
Example:

```
IT-ZIP-CODE DUPLICATED ASCENDING
IT-CITY     DUPLICATED ASCENDING
IT-CUSTOMER-CODE UNIQUE DESCENDING
```

LAYOUT
Contains free-form text, and must therefore be the last clause in the member. It is used to establish the exact layout required for the map, such as the positions of protected and unprotected fields.

Syntax



where

report is the name of a REPORT member

form is the name of a FORM member

group is the name of a GROUP member

item is the name of an ITEM member

data-element is the name of a DATAFLOW, ITEM or GROUP member

common clauses are any of the clauses available to all member types.

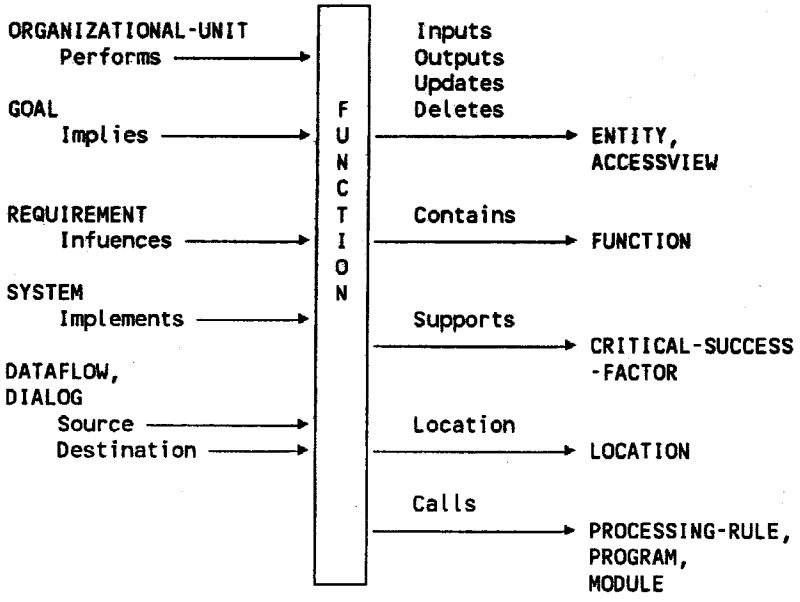
free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 256 characters

Refer to Appendix 2 for details of the common clauses.

FUNCTION

At the SIP level, a function is defined as groups of logically related decisions and activities required to manage a business. This is refined progressively, and at more detailed levels becomes the processing of input information in accordance with established rules to produce output information.
A function which is not divided any further is an elementary function.

The more commonly used relationships to and from FUNCTION are:



INPUTS
Contains the names of any ENTITY and ACCESSVIEW members which input information to this member for processing or reference. It should also list the names of members updated by generation of a new version, rather than replacement of an existing version.

OUTPUTS
Lists the names of members to which this function writes information used for processing or reference. It should also list the names of members which are updated by generating a new version, rather than replacing an existing version.

UPDATES
The only ENTITY members named here are those which this function updates by replacing the original version. Members which are updated by generation of a new version should be declared in the INPUTS and OUTPUTS clauses of the function.

DELETES
This clause may contain the names of one or more ENTITY members, which are deleted by this function.

CALLS
The PROCESSING-RULE, PROGRAM and MODULE members named in this clause are called when the function is performed. This information can be used to find out which processes are common to different functions.

CONTAINS
Must contain the name of every function which is a part of this function.

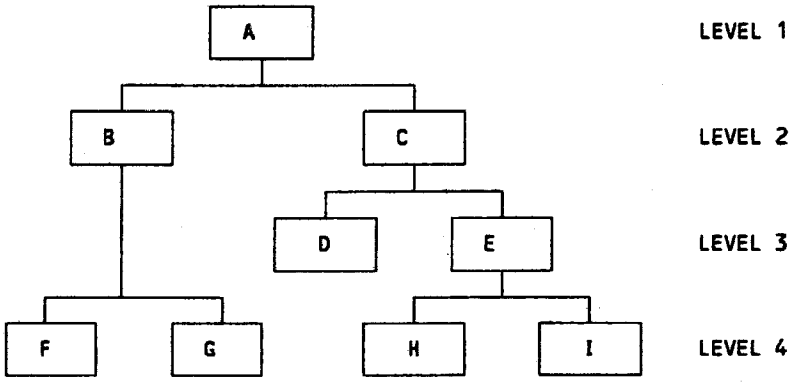
SUPPORTS
Contains the names of one or more critical success factors.

LOCATION
States the location within the enterprise of this function, by naming the appropriate LOCATION members.

ELEMENTARY
This is only specified if the function is elementary; that is, the function is not divided any further.

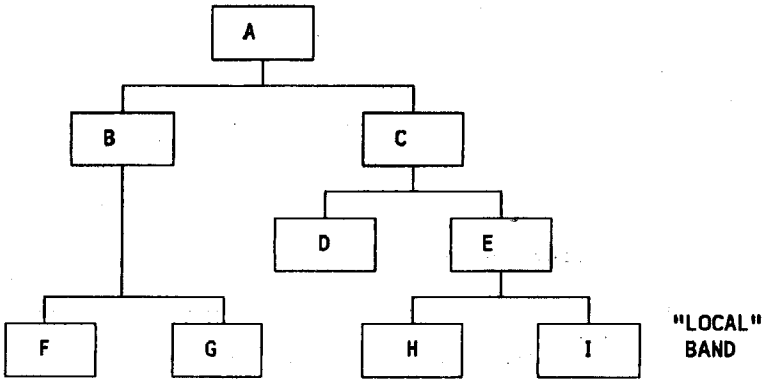
FUNCTION

LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy. The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

BAND
The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.



The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

POSSIBLE-OPERATING-MODE
Contains the name of the possible future operating mode used for the function.

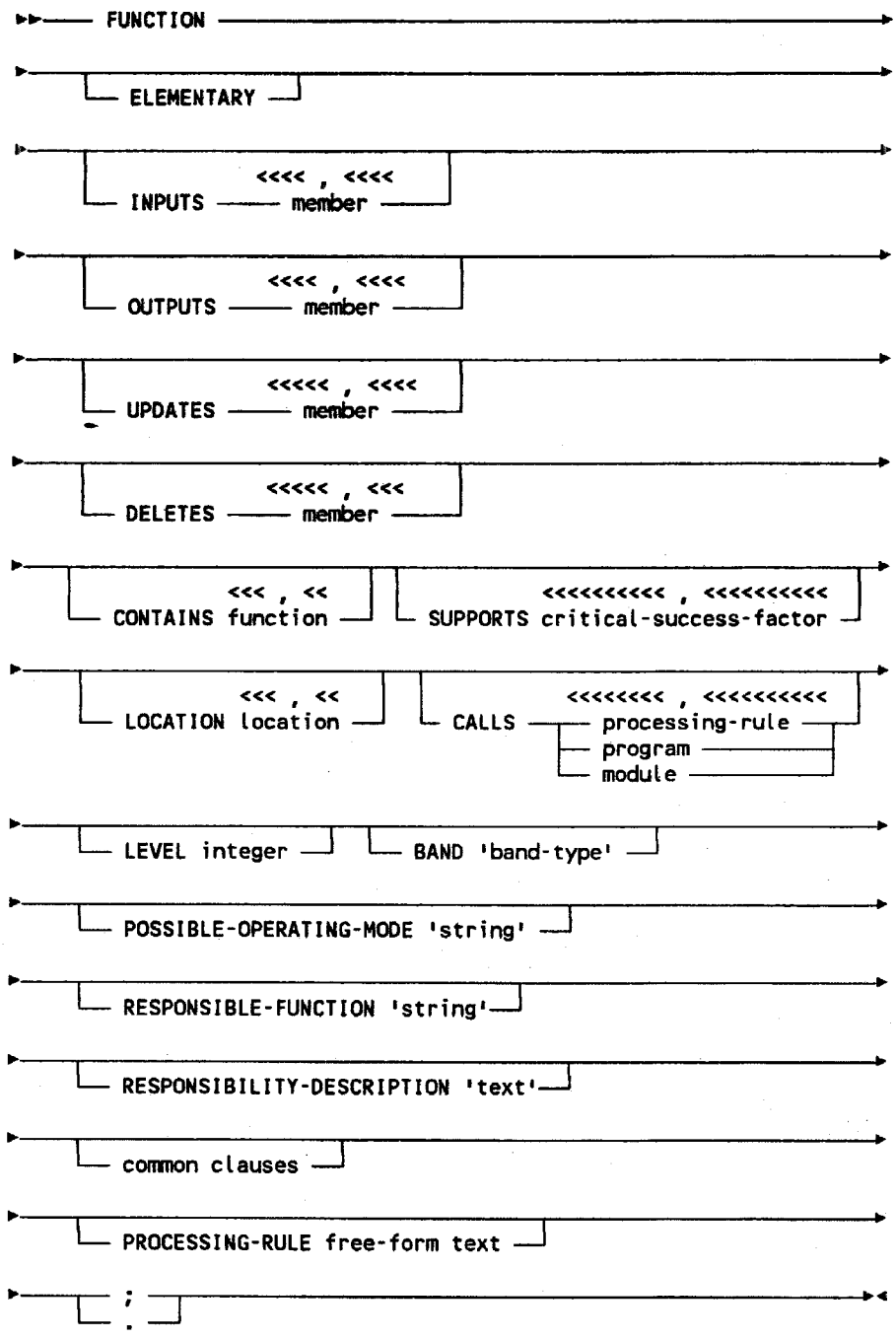
RESPONSIBLE-PERSON
Identifies the person responsible for the performance of the function. It is recommended that the person's job title is used rather than his or her name.

RESPONSIBILITY-DESCRIPTION
Describes the duties and responsibilities of the person named in the RESPONSIBLE-PERSON clause of this FUNCTION member.

PROCESSING-RULE
Contains free-form text without delimiters, and therefore must be the last clause in the member. It stores a procedural description of what the member does, either in text form, or as pseudocode if a COBOL program is to be generated from it.

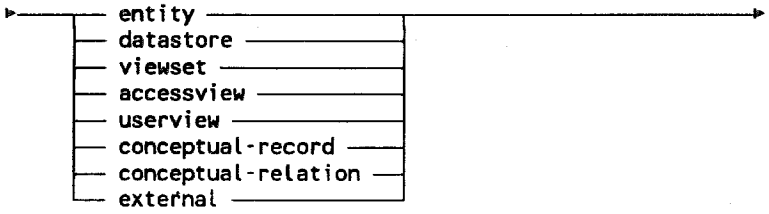
FUNCTION

Syntax



where

member is:



function is the name of a FUNCTION member

critical-success-factor is the name of a CRITICAL-SUCCESS-FACTOR member

location is the name of a LOCATION member

processing-rule is the name of a PROCESSING-RULE member

program is the name of a PROGRAM member

module is the name of a MODULE member

integer is an integer value of up to 18 digits, optionally preceded by a sign

'band-type' is a text string of up to 78 characters, including delimiters.

'string' is a character string of not more than 256 characters

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 248 characters.

Note: the delimiters shown in the above syntax are required when defining a FUNCTION member via the command interface.

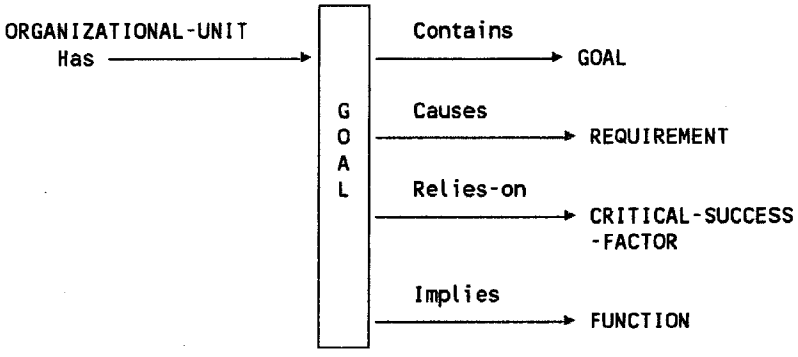
FUNCTION

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

GOAL

A GOAL is the state or position that an organizational unit wishes to attain. It can be permanent (Organizational mission), long-term (Strategic) or short-term (Tactical).
The more commonly used relationships to and from GOAL are:



CONTAINS
Lists earlier goals which make possible the achievement of this goal.

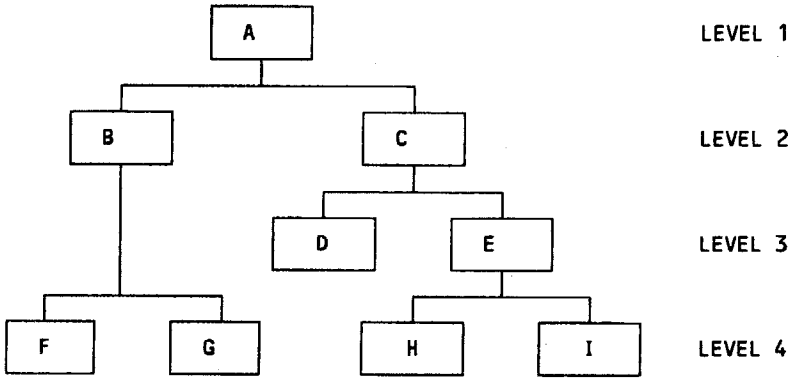
CAUSES
Records the requirements caused by this goal. If a goal does not require some sort of action or change before it can be accomplished, it is probably specified wrongly.

RELIES-ON
Specifies the CSFs upon which the successful attainment of the goal depends.

IMPLIES
A goal is achieved by the fulfilment of requirements which build towards it. The functions implied in the requirements can be recorded in this clause.

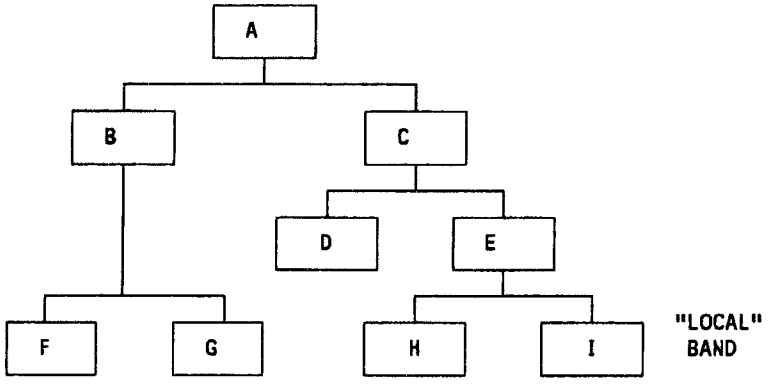
GOAL

LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy.
The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

BAND
The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.



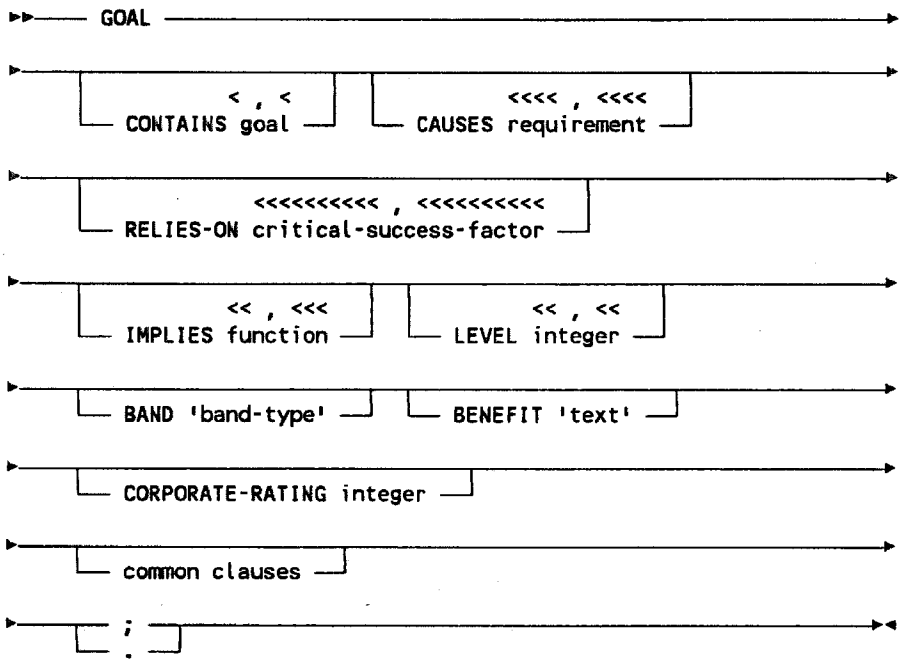
The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

BENEFIT
Records the estimated benefit (in dollars) for the enterprise if the goal is achieved. It may be affected by the estimated cost of the system, or by factors such as weaknesses in the systems involved.

CORPORATE-RATING
Stores an integer reflecting the value to the enterprise of achieving the goal. Any scale of values can be used, but it must be used for every goal in the enterprise.

GOAL

Syntax



where

goal is the name of a GOAL member

requirement is the name of a REQUIREMENT member

critical-success-factor is the name of a CRITICAL-SUCCESS-FACTOR (CSF) member

function is the name of a FUNCTION member

integer is an integer value of up to 18 digits, optionally preceded by a sign

' **band-type**' is a text string of up to 78 characters, including delimiters.

' **text**' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters

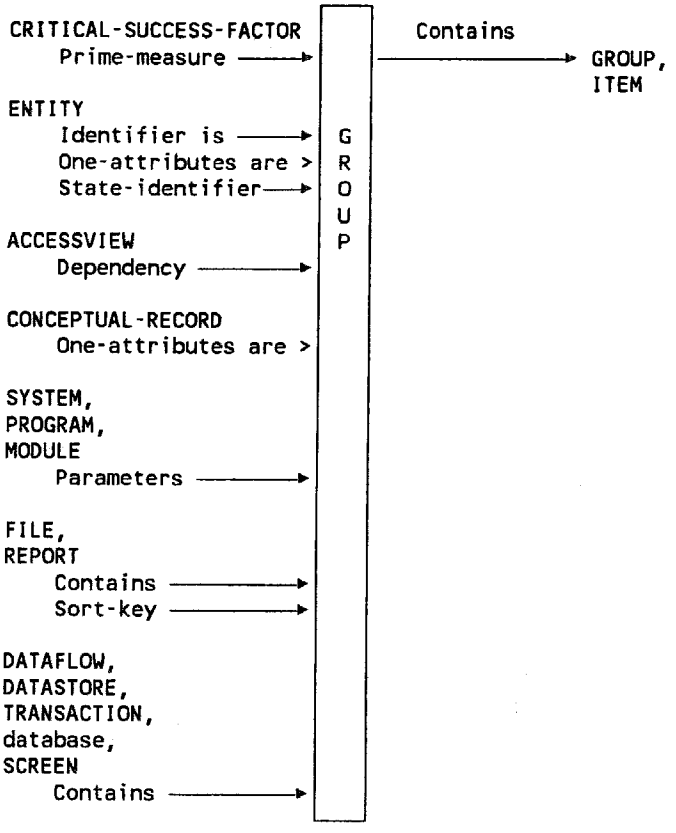
Note: the delimiters shown in the above syntax are required when defining a GOAL member via the command interface.

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

GROUP

A GROUP is an ordered combination of items and/or other groups. The more commonly used relationships to and from GROUP are:



HELD-AS
This clause is used if the member contains items and groups in the HELD-AS form. It is an alternative to ENTERED-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

ENTERED-AS
This clause is used if the member contains items and groups in the ENTERED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

REPORTED-AS
This clause is used if the member contains items and groups in the REPORTED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

DEFAULTED-AS
This clause is used if the member contains items and groups in the DEFAULTED-AS form. It is an alternative to HELD-AS, REPORTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

ALIGNED
This clause is used if all the members in the CONTAINS clause are ALIGNED. It is an alternative to UNALIGNED or NOT-ALIGNED. Only one such clause is valid in the definition.

UNALIGNED
This clause is used if all the members in the CONTAINS clause are UNALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

NOT-ALIGNED
This clause is used if all the members in the CONTAINS clause are NOT-ALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

CONTAINS
Lists the names of GROUP and ITEM members which are part of this member.

For full details of the use of these clauses, refer to the MSP publication "Dictionary/Repository User's Guide".
Examples:

To indicate that the group consists of 3 items:

```
IT-NAME
IT-ADDRESS
IT-ZIP-CODE
```

GROUP

To specify the particular version of component items or groups:

```
IT-NAME 3
IT-ADDRESS  3
```

To indicate that the group consists of 12 occurrences of the same item:

```
(12) IT-MONTH-TOTAL
```

To show the name of a COBOL index for the above:

```
(12) IT-MONTH-TOTAL INDEXED-BY IT-MONTH
```

To indicate that the component items or groups are aligned on the appropriate word boundaries:

```
(12) IT-MONTH-TOTAL ALIGNED
```

To express a COBOL OCCURS DEPENDING-ON:

```
(IT-TOTAL-TIMES) IT-MONTH-TOTAL
```

KEYS
Key fields of the group.
Example:

```
IT-ZIP-CODE DUPLICATED ASCENDING
IT-CITY     DUPLICATED ASCENDING
IT-CUSTOMER-CODE UNIQUE DESCENDING
```

USER-EXIT
Contains the name of a module by which this group is always processed, such as a validation module.

OCCURRENCE
To specify the expected number of occurrences of the relationship, enter:

```
OCCURRENCE integer
```

integer is a positive integer within the OCCURRENCE range.

MAXIMUM-OCCURRENCE
To specify the maximum number of occurrences of the relationship, enter:

```
MAXIMUM-OCCURRENCE integer
```

integer is a positive integer.

MINIMUM-OCCURRENCE
To specify the minimum number of occurrences of the relationship, enter:

```
MINIMUM-OCCURRENCE integer
```

integer is a positive integer.

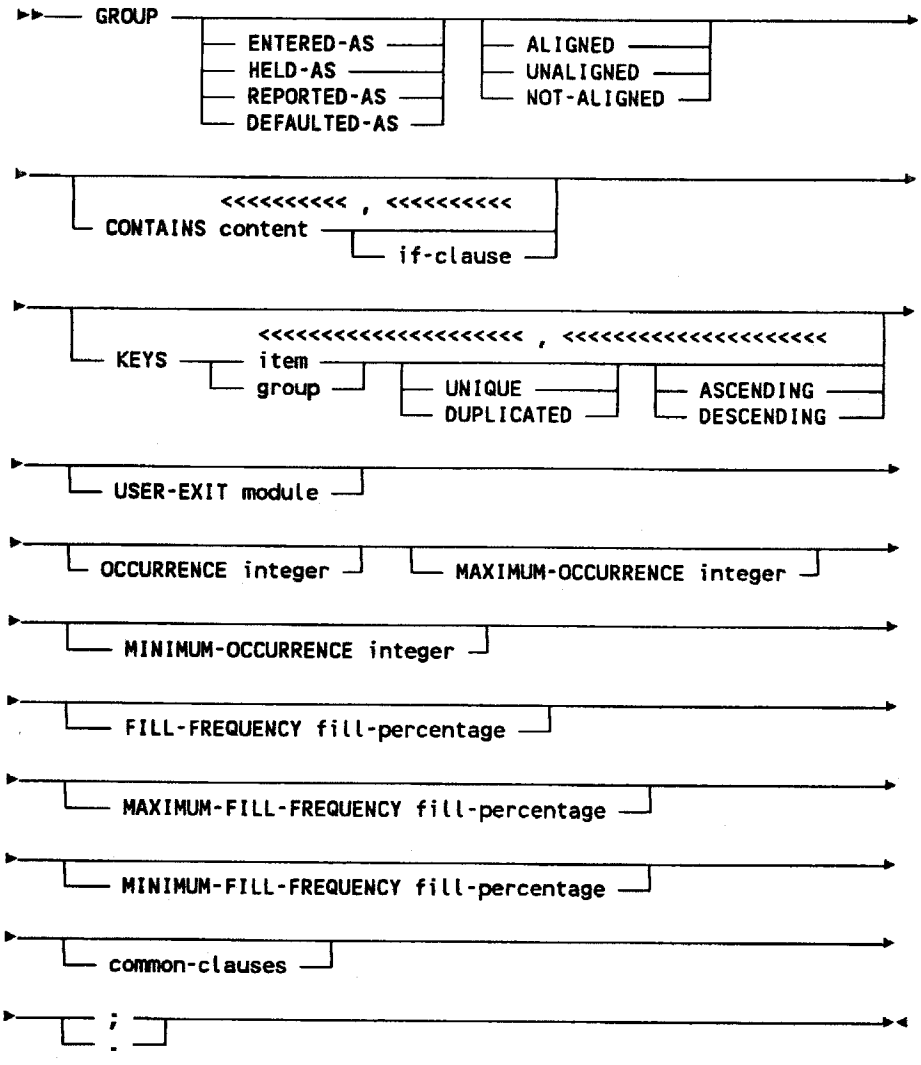
FILL-FREQUENCY
Records the anticipated average percentage of times that this group will contain values. It is relevant to groups which are not mandatory, but which may occur frequently.

MAXIMUM-FILL-FREQUENCY
Records the anticipated maximum percentage of times that this group will contain values. It is relevant to groups which are not mandatory, but which may occur frequently.

MINIMUM-FILL-FREQUENCY
Records the anticipated minimum percentage of times that this group will contain values. It is relevant to groups which are not mandatory, but which may occur frequently.

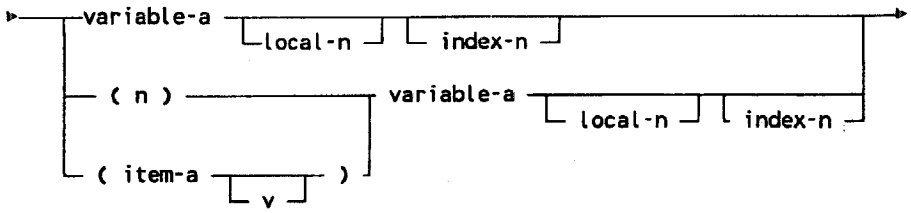
GROUP

Syntax



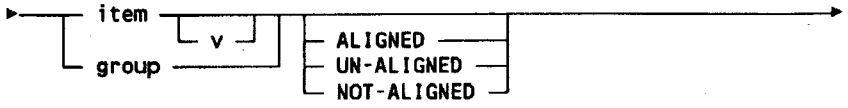
where

content is one of:



where

variable-a is:



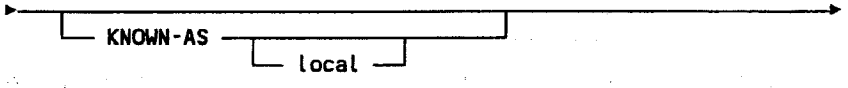
where

item is the name of an ITEM member

group is the name of a GROUP member

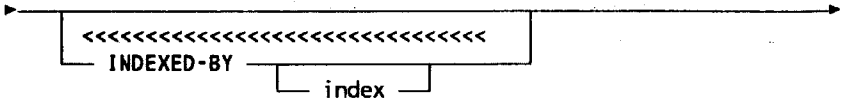
v is an integer in the range 1 to 15. V is not allowed with the keyword KEYS.

local-n is:



where **local** is an alternative name for an ITEM or GROUP

index-n is:

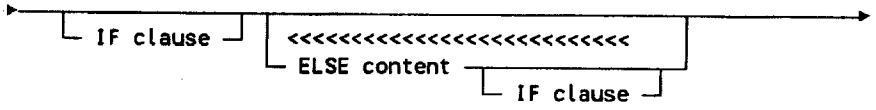


where **index** is the name used as the index name when COBOL data descriptions are generated by Source Language Generation

n is an unsigned integer from 1 to 999,999,999

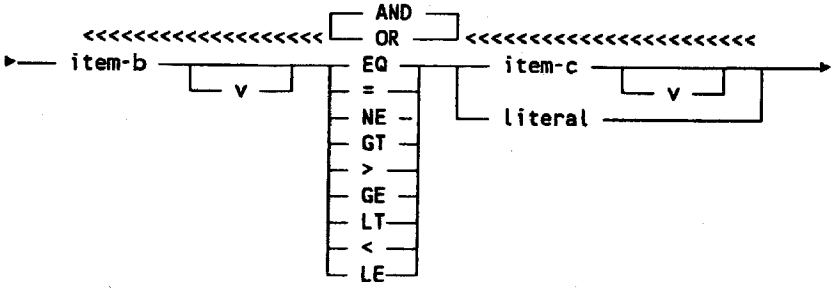
item-a is the name of an ITEM member

if-clause is:



where

clause is:



where

item-b	is the name of an ITEM to be compared with item-c or a literal
v	is as defined above
item-c	is the name of the ITEM to be compared with item-b
literal	is the literal being compared with item-b
EQ or =	means equal to
NE	means not equal to
GT or >	means not greater than
GE	means greater than or equal to
LT or <	means less than
LE	means less than or equal to

content is as defined above

group is as defined above

item is as defined above

module the name of a MODULE member

integer is an integer value of up to 18 digits, optionally preceded by a sign

fill-percentage is an integer in the range 1 to 100

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

HAS-ATTRIBUTES

You use a HAS-ATTRIBUTES member to define a has-attributes relationship. A has-attributes relationship associates an attribute with a business entity or business relationship.

For details of has-attributes relationships refer to the METHODMANAGER Administration manual (MMR-ADMIN).

SOURCE
To specify the business entity or business relationship, enter:

```
SOURCE name
```

name is the name of an ENTITY member or BUSINESS-RELATIONSHIP member.

TARGET
To specify the attribute, enter:

```
TARGET name
```

name is the name of an ITEM member or GROUP member.

ATTRIBUTE-MANDATORY
To specify that the attribute is mandatory, enter:

```
ATTRIBUTE-MANDATORY M
```

To specify that the attribute is optional, enter:

```
ATTRIBUTE-MANDATORY O
```

If the ATTRIBUTE-MANDATORY clause is omitted, optional is assumed.

MINIMUM-CARDINALITY and MAXIMUM-CARDINALITY
To specify a cardinality of p,q enter:

```
MINIMUM-CARDINALITY p
MAXIMUM-CARDINALITY 'q'
```

p and **q** are positive integers and q > = p >0.

To specify a cardinality of many, an unrestricted cardinality, enter:

```
MAXIMUM-CARDINALITY 'm'
```

To specify a cardinality of p,many, enter:

```
MINIMUM-CARDINALITY p
```

p is a positive integer.

If both clauses are omitted, a cardinality of many is assumed.

The MAXIMUM-CARDINALITY clause is a text clause so that the non-specific m for many can be used.

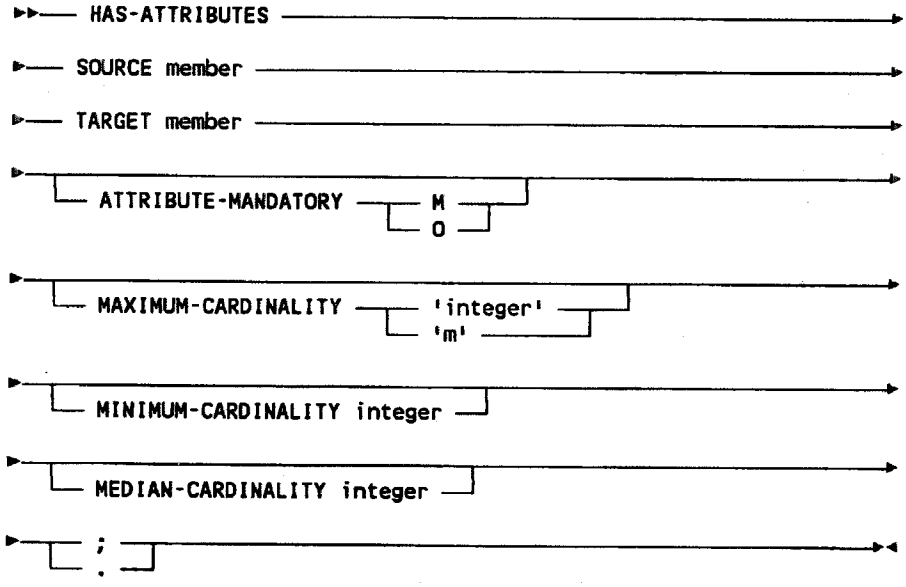
MEDIAN-CARDINALITY
To specify a median cardinality, enter:

```
MEDIAN-CARDINALITY int
```

int is a positive integer within the cardinality range.

HAS-ATTRIBUTES

Syntax



where:

member is the name of a repository member

integer is a positive integer

m is a literal, not a variable.

Note: the delimiters shown in the above syntax are required when defining a HAS-ATTRIBUTES member via the command interface.

IDMS

For details of the IDMS interface member types:

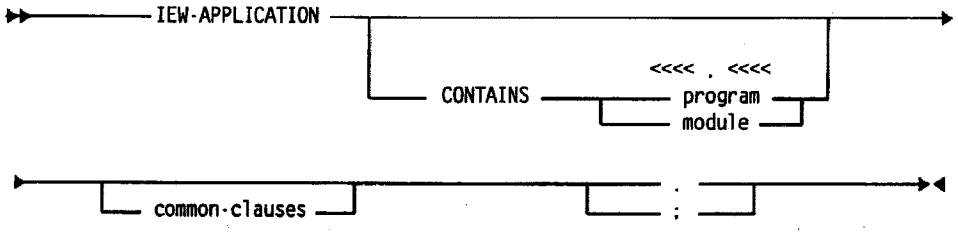
IDMS-AREA
IDMS-DATABASE
IDMS-LOGICAL-RECORD
IDMS-PATH-GROUP
IDMS-RECORD
IDMS-SET
IDMS-SUBSCHEMA
IDMS-VIEW.

please refer to the IDMS/R Interface manual (DMR-IDMSR).

IEW-APPLICATION

ADW/IEW Design Workstation APPLICATION member type.

Syntax



where

program is an IEW-PROGRAM member name

module is an IEW-MODULE member name

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

IEW-ATTRIBUTE-TYPE

ADW/IEW Analysis Workstation ATTRIBUTE-TYPE member type. A type of characteristic or property describable in terms of a value that entities of a given type can have.

CONTAINS
Must contain the name of every GLOBAL-DATA-TYPE or LOCAL-DATA-TYPE which forms part of this ATTRIBUTE-TYPE.

HAS
This clause contains the names of the ATTRIBUTE-TYPES or RELATIONSHIP-TYPES that constitute this ATTRIBUTE-TYPE.

SUPPORTS
Contains the name of every ENTITY-TYPE which this ATTRIBUTE-TYPE supports.

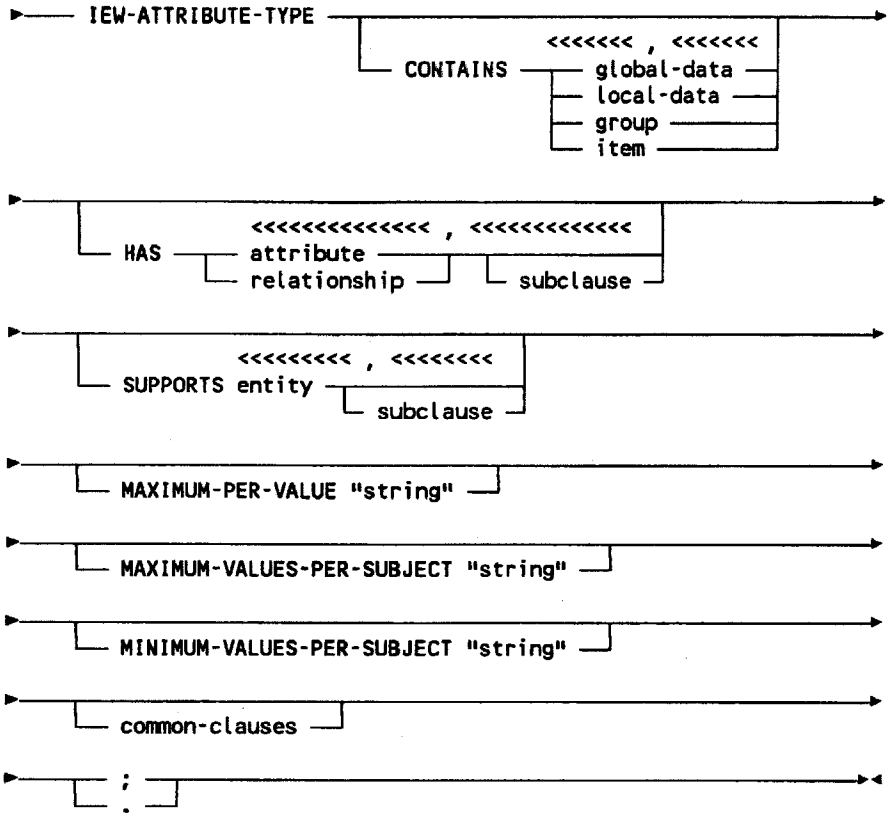
MAXIMUM-PER-VALUE
An integer from 1-9999 or M (for "many")
The maximum number of instances of the described entity type that can be described with each value that is legal for this attribute type. If this value is "1" and if every instance of the described entity must have at least one instance (that is, mandatory) of this attribute type, then the attribute type is a unique identifier.

MAXIMUM-VALUES-PER-SUBJECT
An integer from 1-9999 or M (for "many")
The maximum number of instances of this attribute type that can exist for each entity of the type described by the attribute type. A value of "1" indicates that only one instance of the attribute type is allowed to describe the subject entity.

MINIMUM-VALUES-PER-SUBJECT
An integer from 0-9999
The minimum number of instances of this attribute type that must exist for each entity of the type described by the attribute type. A value of "0" indicates that the attribute type is an optional characteristic of the entity type; a value of "1 indicates that at least one instance of the attribute type is mandatory for describing the subject entity.

IEW-ATTRIBUTE-TYPE

Syntax



where

global-data is an IEW-GLOBAL-DATA-TYPE member name

local-data is an IEW-LOCAL-DATA-TYPE member name

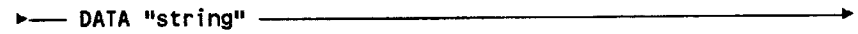
group is a GROUP member name

item is an ITEM member name

attribute is an IEW-ATTRIBUTE-TYPE member name

relationship is an IEW-RELATIONSHIP-TYPE member name

subclause is



where **"string"** is the data for import from, or export to, ADW/IEW

entity is an IEW-ENTITY-TYPE member name

"string " is as defined above

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-CRITICAL-ASSUMPTION

ADW/IEW Planning Workstation CRITICAL-ASSUMPTION member type. An assumption held about the enterprise, business environment, competition or industry that supports or validates a critical success factor.

CAUSES
Records the problems caused by this CRITICAL-ASSUMPTION.

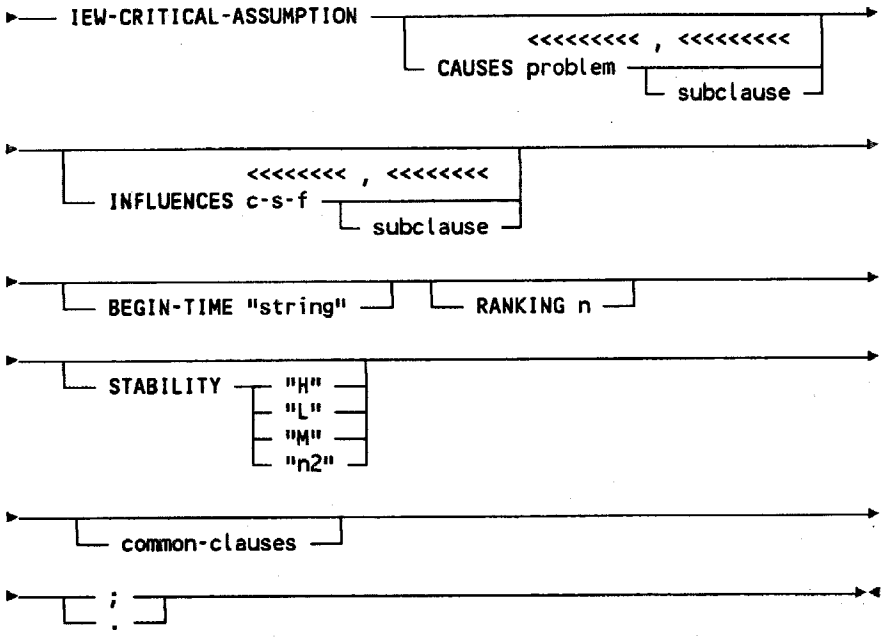
INFLUENCES
Specifies the CRITICAL-SUCCESS-FACTORS impacted by this CRITICAL-ASSUMPTION.

BEGIN-TIME
Date from which planning entity is effective
in the form MMM-YYYY.
The past, present or future date a critical assumption is initially effective.

RANKING
Relative importance of Planning Object.
An integer from 1-9999
How important is this critical assumption relative to other critical assumptions of the enterprise?

STABILITY
Degree of certainty that this PLanning Object will remain stable over time.
What degree of certainty is there that this critical assumption will remain stable over time?
H - high
M - medium
L - low, or an integer from 1-999
Values H(igh) M(edium) L(ow) or integer from 1 - 999

Syntax

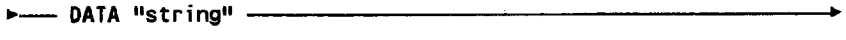


where

problem is an IEW-PROBLEM member name

c-s-f is an IEW-CRITICAL-SUCCESS-FACTOR member name

subclause is



where "**string**" is the data for import from, or export to, ADW/IEW

"**string**" is as defined above

n is an unsigned numeric value in the range 1 to 9999

n2 is an unsigned numeric value in the range 1 to 999

common-clauses are any of the clauses available to all member types. Refer to Appendix 2 for details of the common clauses.

IEW-CRITICAL-SUCCESS-FACTOR

ADW/IEW Planning Workstation CRITICAL-SUCCESS-FACTOR.
A fundamental aspiration of the enterprise whose achievement
is necessary to meet other goals and objectives.

CAUSES
Records the problems caused by this CRITICAL-SUCCESS-FACTOR.

INFLUENCES
Specifies the GOALS impacted by this CSF.

BEGIN-TIME
Date from which planning entity is effective
in the form MMM-YYYY.
The past, present or future date a critical assumption is
initially effective.

RANKING
Relative importance of Planning Object.
An integer from 1-9999
How important is this critical assumption relative to other
critical assumptions of the enterprise?

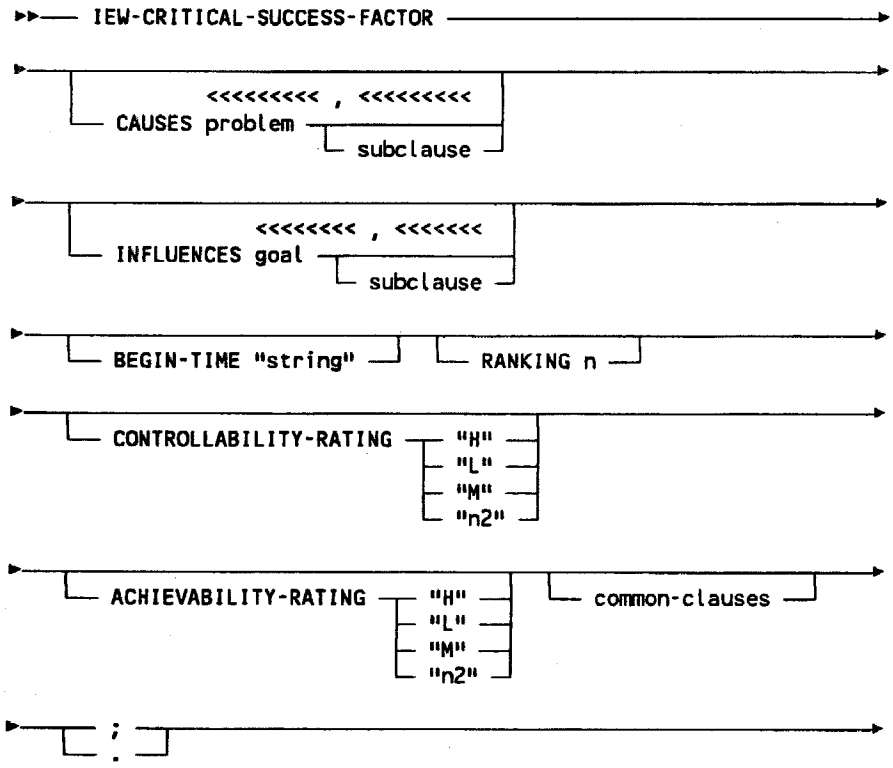
CONTROLLABILITY-RATING
Degree of certainty that this Crtical Success Factor
can be controlled.
What degree of certainty is there that this critical success
factor can be controlled?
H - high
M - medium
L - low, or an integer from 1-999

ACHIEVABILITY-RATING
Degree of certainty that this Crtical Success Factor
can be achieved.
What degree of certainty is there that this critical success
factor can be achieved?

H - high
M - medium
L - low, or an integer from 1-999
Values H(igh) M(edium) L(ow) or integer from 1 - 999

IEW-CRITICAL-SUCCESS-FACTOR

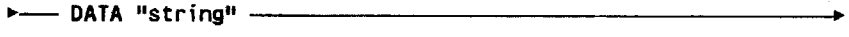
Syntax



where

problem is an IEW-PROBLEM member name

subclause is



where "**string**" is the data for import from, or export to, ADW/IEW

goal is an IEW-GOAL member name

"**string**" is as defined above

n is an unsigned number in the range 1 to 9999

n2 is an unsigned number in the range 1 to 999

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-DATA-COLLECTION

ADW/IEW Planning Workstation DATA-COLLECTION member-type.
A collection of data that is implemented as a unit, which contains facts about entities of interest to the enterprise.

CAUSES
Records the problems caused by this DATA-COLLECTION.

IMPLEMENTS
Contains the name of each ENTITY-TYPE or SUBJECT-AREA implemented by this DATA-COLLECTION.

SEE
List of ENTITY-TYPES, ATTRIBUTE-TYPES or RELATIONSHIP-TYPES that are involved in this DATA-COLLECTION.

SUPPORTS
Must contain the name of every planning object which is supported by this data-collection.

HAS
This clause contains the names of the DATA-COLLECTIONS that constitute this DATA-COLLECTION.

AFFECTS
Contains the names of any data-collections which are affected by this data-collection.

ACCESSIBILITY-RATING
Degree of accessibility of Data Collection.
Values H(igh) M(edium) L(ow) or integer from 1 - 999
How accessible is this data collection? Was it designed to meet a specific information need only or to support flexible access strategies?
H - high
M - medium
L - low, or an integer from 1-999

INTEGRITY-RATING
Quality/Integrity rating, including the degree of monitoring and validation.
Values H(igh) M(edium) L(ow) or integer from 1 - 999

REDUNDANCY-RATING
Degree of replication.
Values H(igh) M(edium) L(ow) or integer from 1 - 999

SECURITY-RATING
Rating in terms of impenetrability, recoverability and auditability.
Values H(igh) M(edium) L(ow) or integer from 1 - 999

DOCUMENTATION-RATING
How well documented is this Planning Object ?
Values H(igh) M(edium) L(ow) or integer from 1 - 999

EFFICIENCY-RATING
Rating of efficiency in terms of performance.
Values H(igh) M(edium) L(ow) or integer from 1 - 999

MAINTAINABILITY-RATING
Rating in terms of simplicity, expandibility and portability.
Values H(igh) M(edium) L(ow) or integer from 1 - 999

DISPOSITION
Indication of what to do with an existing Planning Object.
A character code that indicates what planners have decided to do with an existing data collection.
KE - keep as is
IN - interface, or
RE - replace
KEep as is: The data collection is an acceptable implementation of its associated entity types. Retain it substantially unchanged.
INterface: The data collection needs work to be able to interface with the rest of the information system.
REplace: The data collection should be replaced with a new one.

IEW-DATA-COLLECTION

IN - Interface with rest of system
RE - Replace

DISPOSITION-RATIONALE
Text supporting the decision on DISPOSITION.
Up to five lines of text.
A textual explanation supporting the planners' decision of what to do with an existing object. Especially in cases where a decision as to what to do with an object might be questioned, this property will allow planners to document the reasons why they made that decision.

TECHNOLOGY

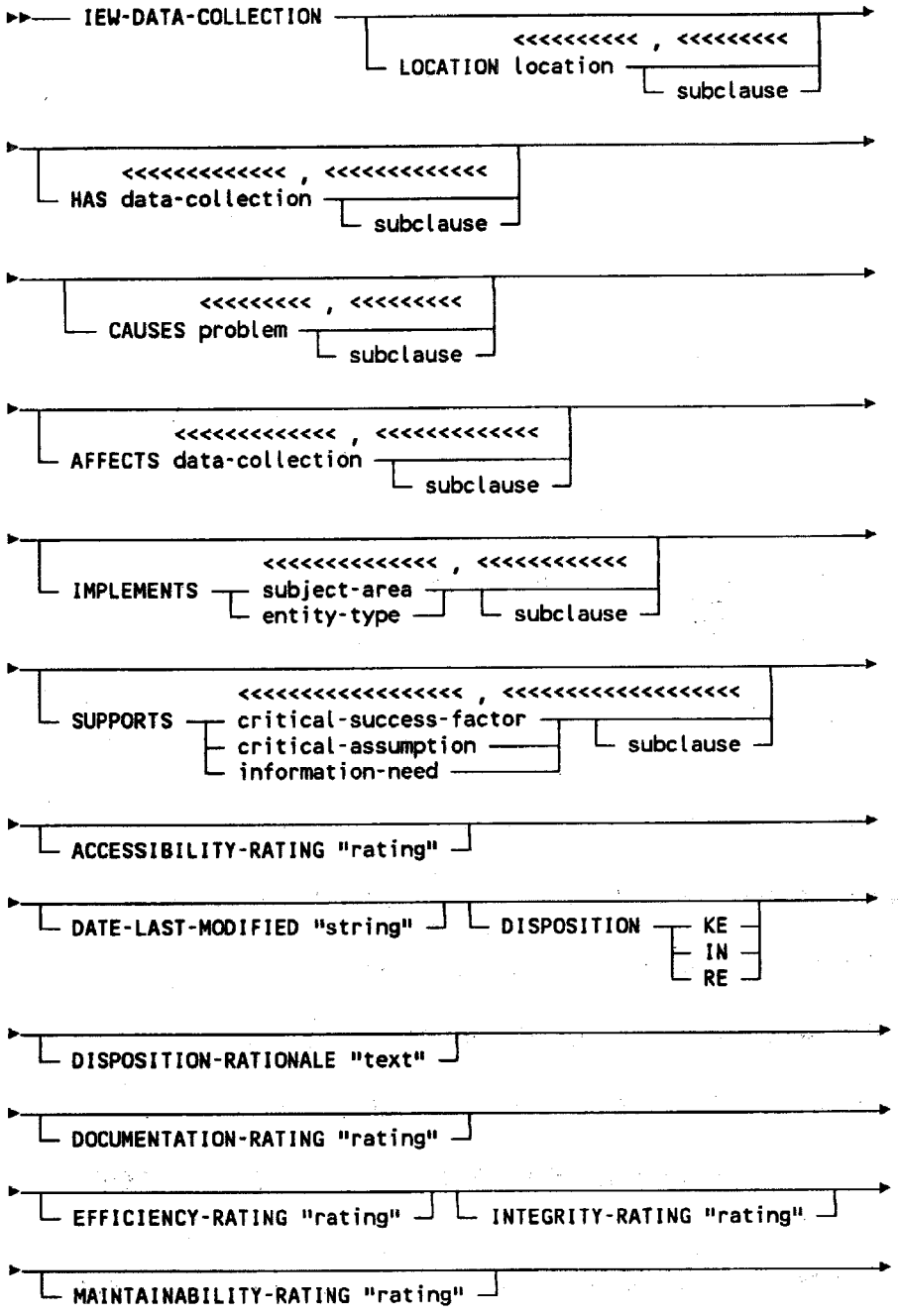
Data Collections	Mechanisms
HI - Hierarchical DBMS	AU - Automated
RE - Relational DBMS	MA - Manual
NT - Network DBMS	BO - Both
DB - DBMS maintained	
FI - Computer files	
MF - Microfiche files	
PA - Paper files	

STATUS
Code used to describe the state of the members types.
The codes have the following meanings:
PL - PLANNED
CU - CURRENT
PA - PAST
UD - UNDER DEVELOPMENT
AA - CREATED BY AFFINITY ANALYSIS

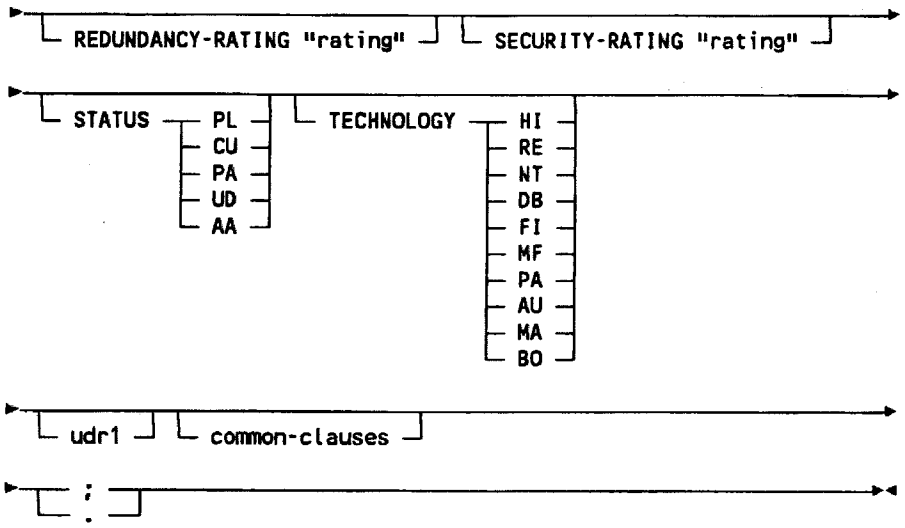
LOCATION
States the location within the enterprise of this object by naming the appropriate LOCATION members.

DATE-LAST-MODIFIED
Date in the form: mm/dd/yyyy
The date of the most recent major design modification of the Data Collection.

Syntax


IEW-DATA-COLLECTION
LOCATION location
subclause
HAS data-collection
subclause
CAUSES problem
subclause
AFFECTS data-collection
subclause
IMPLEMENTS
subject-area
entity-type
subclause
SUPPORTS
critical-success-factor
critical-assumption
information-need
subclause
ACCESSIBILITY-RATING "rating"
DATE-LAST-MODIFIED "string"
DISPOSITION
KE
IN
RE
DISPOSITION-RATIONALE "text"
DOCUMENTATION-RATING "rating"
EFFICIENCY-RATING "rating"
INTEGRITY-RATING "rating"
MAINTAINABILITY-RATING "rating"

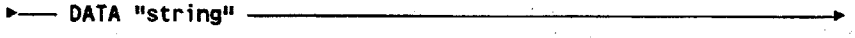
IEW-DATA-COLLECTION



where

location is an IEW-LOCATION member name

subclause is



where

"string" is the data to be imported from, or exported to, ADW/IEW

data-collection is an IEW-DATA-COLLECTION member name

problem is an IEW-PROBLEM member name

subject-area is an IEW-SUBJECT-AREA member name

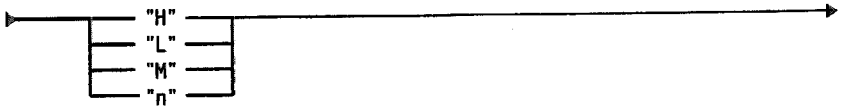
entity-type is an IEW-ENTITY-TYPE member name

critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member name

critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

information-need is an IEW-INFORMATION-NEED member name

"rating" is



"n" is an integer in the range 1 to 999

"string" is as defined above

"text" is up to 32767 delimited strings, each string being a maximum of 60 characters long

udr1 is as defined in Appendix 2.

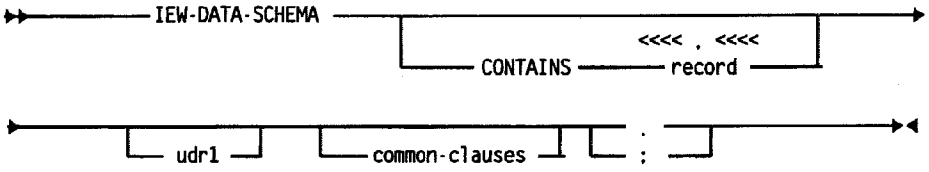
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-DATA-SCHEMA

ADW/IEW Design Workstation DATA-SCHEMA member type.

Syntax



where

record is an IEW-GLOBAL-DATA-RECORD member name

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

IEW-DATA-STRUCTURE-OR-BLOCK

ADW/IEW Design Workstation DATA-STRUCTURE-OR-BLOCK member type.

HELD-AS
This clause is used if the member contains items and groups in the HELD-AS form. It is an alternative to ENTERED-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

ENTERED-AS
This clause is used if the member contains items and groups in the ENTERED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and REPORTED-AS.

REPORTED-AS
This clause is used if the member contains items and groups in the REPORTED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

DEFAULTED-AS
This clause is used if the member contains items and groups in the DEFAULTED-AS form. It is an alternative to HELD-AS, REPORTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

ALIGNED
This clause is used if all the members in the CONTAINS clause are ALIGNED. It is an alternative to UNALIGNED or NOT-ALIGNED. Only one such clause is valid in the definition.

UNALIGNED
This clause is used if all the members in the CONTAINS clause are UNALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

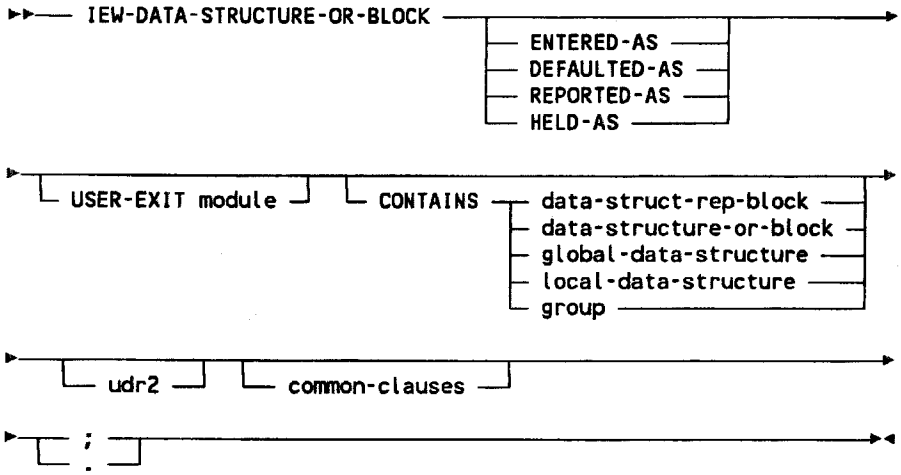
NOT-ALIGNED
This clause is used if all the members in the CONTAINS clause are NOT-ALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

USER-EXIT
Contains the name of a module by which this group is always processed, such as a validation module.

CONTAINS
Lists the names of DATA-STRUCTURE-OR-BLOCKS, DATA-STRUCTURE-REPETITION-BLOCKS, LOCAL-DATA-STRUCTURES or GLOBAL-DATA-STRUCTURES contained by this data-structure.

IEW-DATA-STRUCTURE-OR-BLOCK

Syntax



where

module is an IEW-MODULE member name

data-structure-or-block is an IEW-DATA-STRUCTURE-OR-BLOCK member name

data-struct-rep-block is an IEW-DATA-STRUCT-REP-BLOCK member name

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

group is a GROUP member name

udr2 is as defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-DATA-STRUCT-REP-BLOCK

ADW/IEW Design Workstation DATA-STRUCTURE-REPETITION-BLOCK member type.

HELD-AS
This clause is used if the member contains items and groups in the HELD-AS form. It is an alternative to ENTERED-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

ENTERED-AS
This clause is used if the member contains items and groups in the ENTERED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

REPORTED-AS
This clause is used if the member contains items and groups in the REPORTED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

DEFAULTED-AS
This clause is used if the member contains items and groups in the DEFAULTED-AS form. It is an alternative to HELD-AS, REPORTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

ALIGNED
This clause is used if all the members in the CONTAINS clause are ALIGNED. It is an alternative to UNALIGNED or NOT-ALIGNED. Only one such clause is valid in the definition.

UNALIGNED
This clause is used if all the members in the CONTAINS clause are UNALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

NOT-ALIGNED
This clause is used if all the members in the CONTAINS clause are NOT-ALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

SYNCHRONISED-INDICATOR
Is this object synchronised, indicate Y or N.

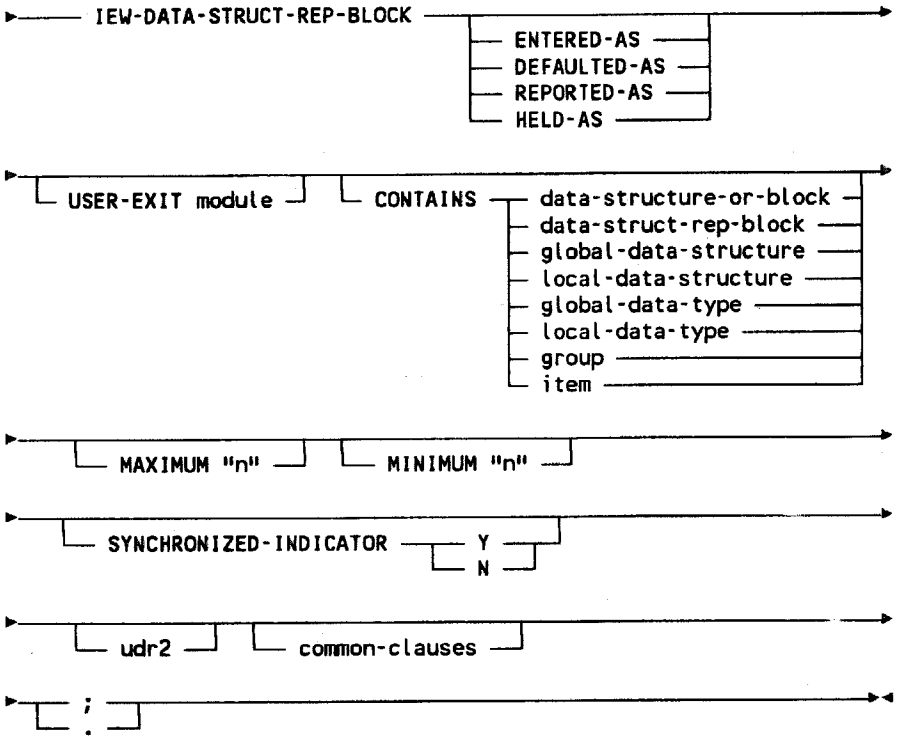
USER-EXIT
Contains the name of a module by which this group is always processed, such as a validation module.

CONTAINS
Lists the names of DATA-STRUCTURE-OR-BLOCKS, DATA-STRUCTURE-REPETITION-BLOCKS, LOCAL-DATA-STRUCTURES or GLOBAL-DATA-STRUCTURES contained by this data-structure.

MAXIMUM
Maximum value applicable.

MINIMUM
Minimum value applicable.

Syntax



where

module is an IEW-MODULE member name

data-structure-or-block is an IEW-DATA-STRUCTURE-OR-BLOCK member name

data-struct-rep-block is an IEW-DATA-STRUCT-REP-BLOCK member name

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

IEW-DATA-STRUCTURE-REP-BLOCK

global-data-type is an IEW-GLOBAL-DATA-TYPE member name

local-data-type is an IEW-LOCAL-DATA-TYPE member name

group is a GROUP member name

"n" is an integer in the range 1 to 32767

udr2 is as defined in Appendix 2

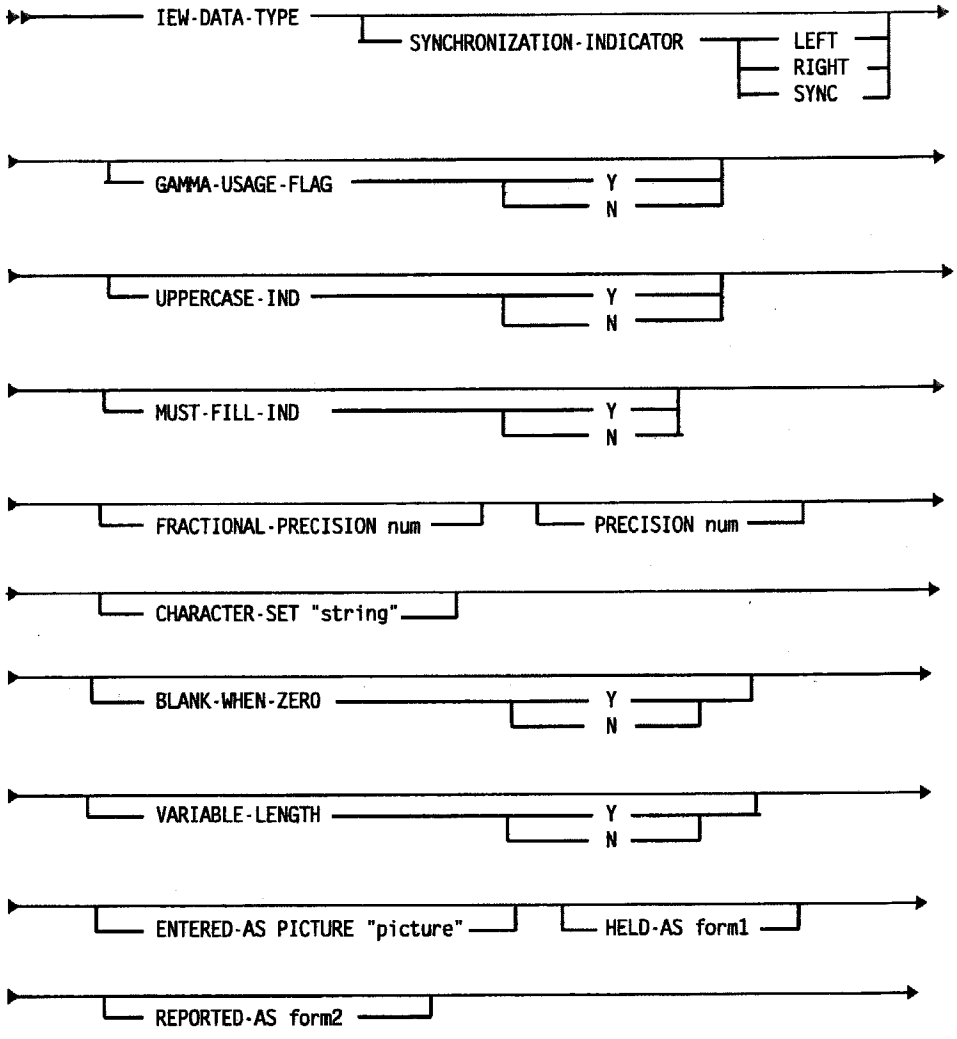
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

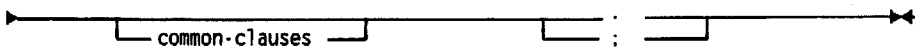
IEW-DATA-TYPE

ADW/IEW Design Workstation DATA-TYPE member type.

Syntax



IEW-DATA-TYPE



where

num is a small unsigned integer

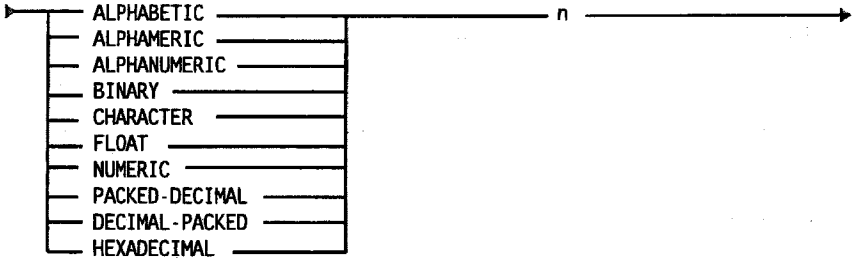
"**string**" is data to be imported from, or exported to, ADW/IEW

"**picture**" is a valid data-type format

common-clauses are any type of the clauses available to all member types

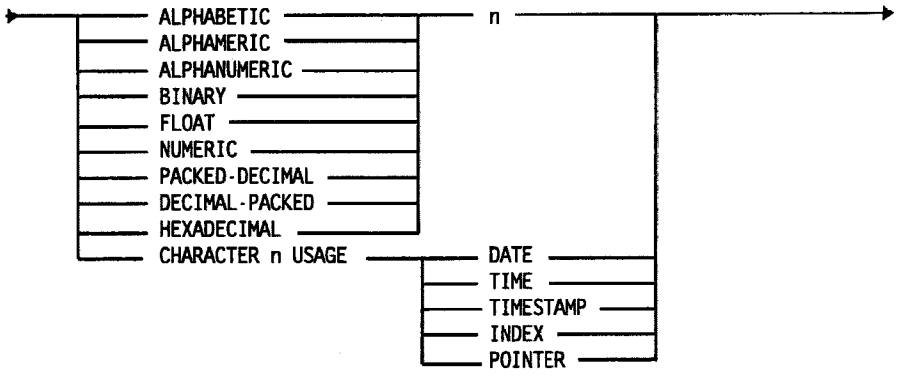
Refer to Appendix 2 for details of the common clauses.

form1 is



where **n** is an unsigned decimal number specifying a maximum length

form2 is

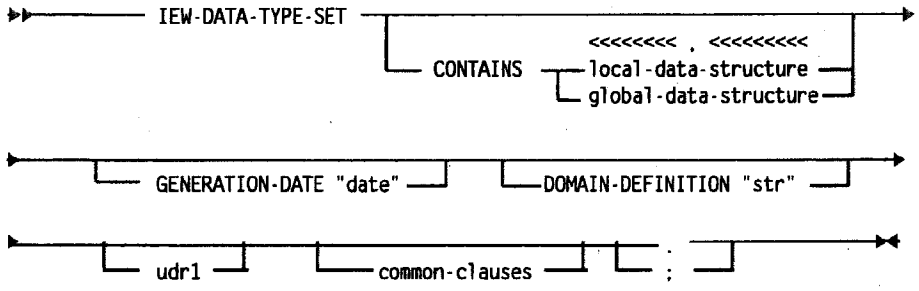


where **n** is as defined above.

IEW-DATA-TYPE-SET

ADW/IEW Design Workstation DATA-TYPE-SET member type.

Syntax



where

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

"date" is a string in date format

"str" is data to be imported from, or exported to, ADW/IEW

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-DATAFLOW

ADW/IEW Analysis Workstation DATAFLOW member type.
A named package of data objects (entity types, attribute types and relationship types) or other data flows that are passed between data flow nodes. (Processes, data stores, external agents and junctions are all data flow nodes).

SOURCE
Must name the source of the dataflow, which may be one of the following;
PROCESS
SEQUENTIAL-PROCESS
DATASTORE
EXTERNAL-AGENT
JUNCTION
(There can only be one source for any one dataflow).

DESTINATION
Must name the destination of the dataflow, which may be one of the following;
PROCESS
SEQUENTIAL-PROCESS
DATASTORE
EXTERNAL-AGENT
JUNCTION
(There can only be one destination for any one dataflow).

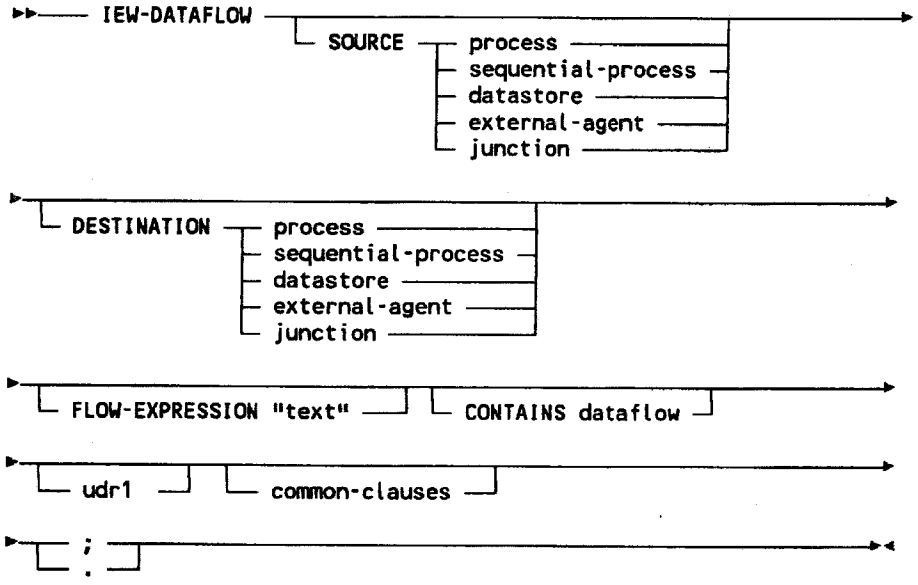
CONTAINS
Must contain the name of every DATAFLOW which constitutes this DATAFLOW.

SEE
List of ENTITY-TYPES, ATTRIBUTE-TYPES or RELATIONSHIP-TYPES that are involved in this DATAFLOW.

FLOW-EXPRESSION
Used to specify the components of the dataflow.

IEW-DATAFLOW

Syntax



where

process is an IEW-PROCESS member name

sequential-process is an IEW-SEQUENTIAL-PROCESS member name

datastore is an IEW-DATASTORE member name

external-agent is an IEW-EXTERNAL-AGENT member name

junction is an IEW-JUNCTION member name

"text" is up to 32767 delimited strings, each string being a maximum of 60 characters long

dataflow is an IEW-DATAFLOW member name

udr1 is as defined in Appendix 2

common-clauses are any of the clauses available to all member types.

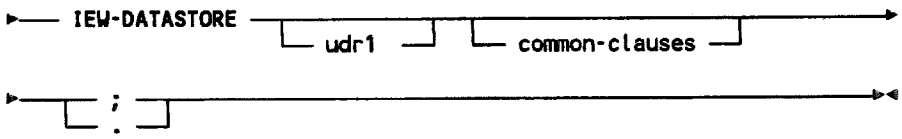
Refer to Appendix 2 for details of the common clauses.

IEW-DATASTORE

ADW/IEW Analysis Workstation DATASTORE member-type.
An arrangement by which data produced by one process is preserved over time and made available to other processes or subsequent executions of the same process.

SEE
List of ENTITY-TYPES, ATTRIBUTE-TYPES or RELATIONSHIP-TYPES that are involved in this DATASTORE.

Syntax



where

udr1 is as defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

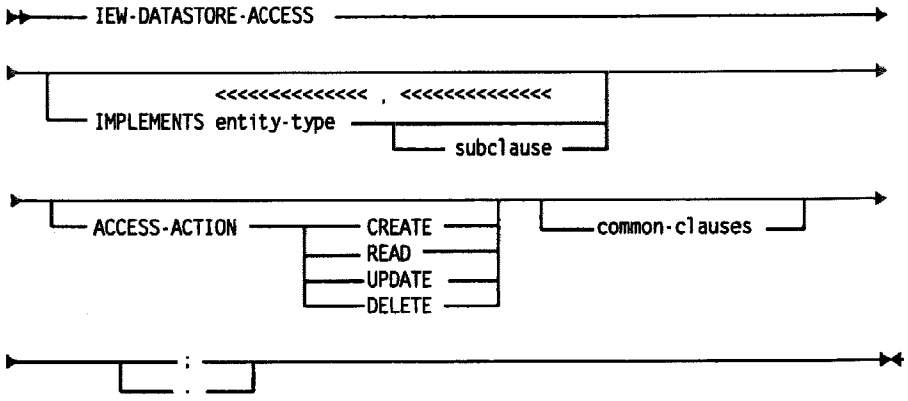
IEW-DATASTORE-ACCESS

ADW/IEW Analysis Workstation DATASTORE-ACCESS member type.

IMPLEMENTS
A list of ENTITY-TYPES accessed by this DATA-STORE-ACCESS.

ACCESS-ACTION
Type of access required where values allowed are:
CREATE, READ, UPDATE, DELETE

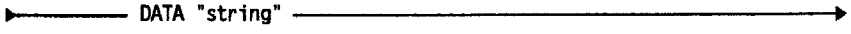
Syntax



where

entity-type is an IEW-ENTITY-TYPE member name

subclause is



where **"string"** is the data to be imported from, or exported to, ADW/IEW

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-DB2-DATABASE

ADW/IEW Design Workstation DB2-DATABASE member type.
All the clauses available to define IEW-DB2-DATABASE are optional.

AS

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

STOGROUP

To define the storage group to which the database belongs, enter:

```
STOGROUP stogroup-name
```

stogroup-name is the name of an IEW-DB2-STOGROUP member. On encoding, the member specified in the STOGROUP clause is checked to ensure that it is an IEW-DB2-STOGROUP member. The storage group defined in the IEW-DB2-DATABASE definition is the default used by generated table spaces and indexes:

- that belong to the database
- that do not have a storage group specified in their own member definition.

If there is no storage group defined in the IEW-DB2-DATABASE member, none is generated and the DB2 default, SYSDEFLT, applies. However you are recommended to explicitly define the storage group, even if it is SYSDEFLT, so that the repository accurately reflects your DB2 environment.

BUFFERPOOL
To define the buffer pool that databases use, enter:

```
BUFFERPOOL bufferpool-name
```

bufferpool-name is one of the following buffer pools:

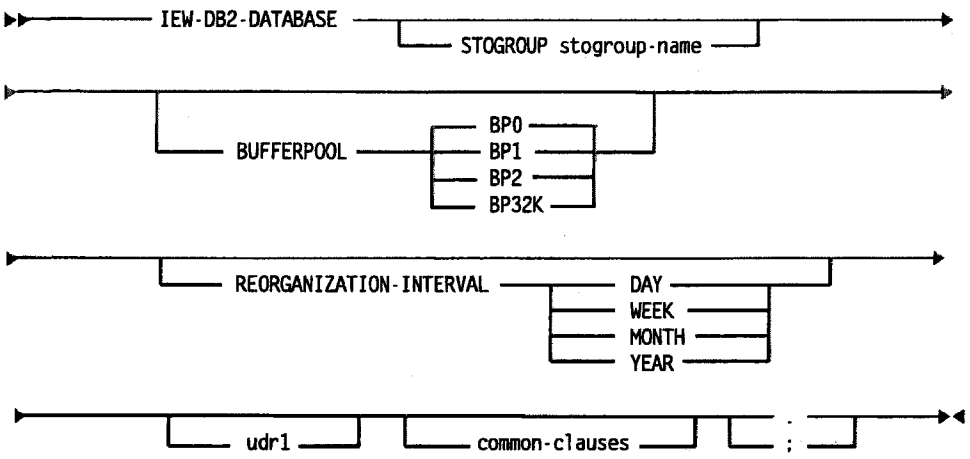
```
BP0             BP1
BP2             BP32K
```

The buffer pool defined in the DB2-DATABASE definition is the default used by those table spaces and indexes that belong to the database, and do not have a buffer pool specified in their own member definition.

If you do not define a buffer pool in a DB2-DATABASE, none is generated and the DB2 default applies.

IEW-DB2-DATABASE

Syntax



where

stogroup-name is the name of an IEW-DB2-STOGROUP member

udr1 is as defined in Appendix 2

common clauses are any of the clauses common to all member types.

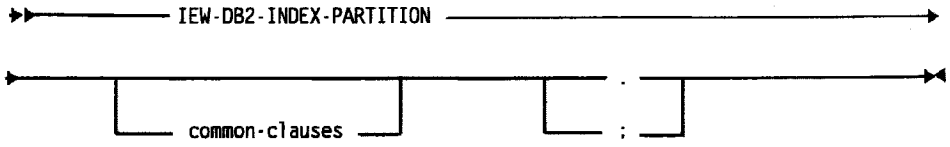
Refer to Appendix 2 for details of the common clauses.

IEW-DB2-INDEX-PARTITION

ADW/IEW Design Workstation DB2-INDEX-PARTITION member type.

Documentary object added to hold additional attributes other than those already held in the PARTITION clause of the related IEW-INDEX.

Syntax



where

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-DB2-STOGROUP

ADW/IEW Design Workstation DB2-STORAGE-GROUP member type.

All the clauses available to define IEW-DB2-STOGROUP members are optional. However, the VOLUMES and VCAT clauses must be present for the successful generation of SQL CREATE STOGROUP statements.

AS
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

VOLUMES
To define the DASD volumes on which the storage group resides, enter:

```
VOLUMES vol-id-list
```

vol-id-list is one or more storage volume names, each of no more than six characters, and separated by commas. You can define a maximum of 133 storage volume names. At least one volume must be defined for the successful generation of an SQL CREATE STOGROUP statement. When you generate SQL statements, DB2 names for volumes are taken directly from the names you specify in this clause.

VCAT
To define a control or master level password used to access the VSAM catalog, enter:

```
VCAT catalog
```

catalog is a VSAM catalog name, of no more than eight characters.
When you generate SQL statements, the VCAT password is taken directly from the password you specify in this clause.

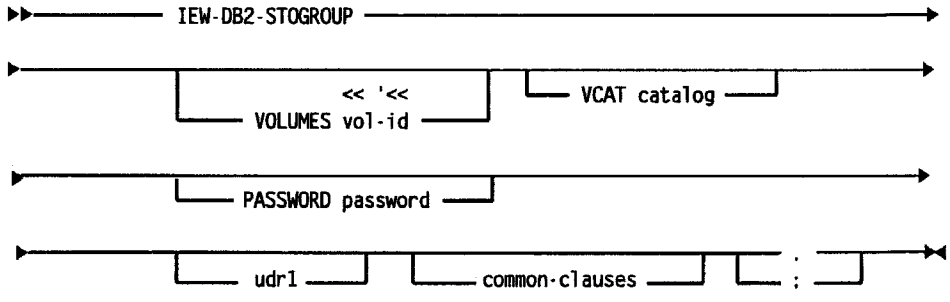
PASSWORD
To define a control or master level password used to access the VSAM catalog, enter:

```
PASSWORD password
```

password is a control or master level VSAM catalog password, of no more than eight characters.

When you generate SQL statements, the VCAT password is taken directly from the password you specify in this clause.

Syntax



where

vol-id is a storage volume name, of no more than 6 characters

catalog is a VSAM catalog name, of no more than 8 characters

IEW-DB2-STOGROUP

password is a VSAM catalog password, of no more than 8 characters

udrl is as defined in Appendix 2

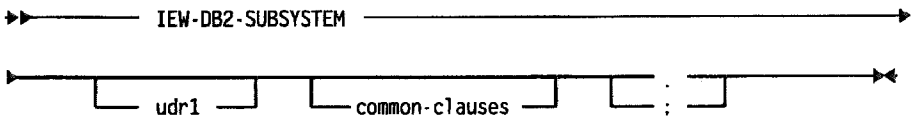
common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-DB2-SUBSYSTEM

ADW/IEW Design Workstation DB2-SUBSYSTEM member type.

Syntax



where

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-DB2-TABLE

ADW/IEW Design Workstation RELATIONAL-TABLE member type.

Tables are of major interest to the end-user, since they contain the user's own data and are not a product of the database administrator or other technical specialist. Therefore the IEW-DB2-TABLE is one of the most used IEW-DB2 member types. All the clauses available to define IEW-DB2-TABLE members are optional. However, for the successful generation of SQL statements you must define specific clauses, as follows:

- for ALTER TABLE statements define the CREATOR-OWNER clause
- for CREATE TABLE statements define the CREATOR-OWNER, COLUMNS and IN clauses
- for COMMENT ON statements define the CREATOR-OWNER and DB2-COMMENT clauses
- for DECLARE TABLE statements define the CREATOR-OWNER and COLUMNS clauses
- for DROP TABLE statements define the CREATOR-OWNER clause
- for LABEL ON statements define the CREATOR-OWNER and DB2-LABEL clauses.

To specify the ITEM and GROUP members that represent the columns of the table, use the CONTAINS clause. It establishes relationships between an IEW-DB2-TABLE and ITEM and GROUP members, which define the table's columns. These GROUPs and ITEMs may also form part of other file and database segment definitions. For example, installations with IMS may already have GROUP and ITEM definitions in the repository, which can now be shared with the DB2 environment.
You can define columns:

- individually, so that one ITEM or GROUP member defines one column
- in sets, so that the same ITEM or GROUP member defines several columns, with identical attributes
- in cascades from a GROUP member, so that every ITEM nested in a GROUP member defines one column.

Sub-clauses within the COLUMNS clause enable you to define:

- the names of columns
- whether or not the column can contain a null value
- the attributes of the column, for example if it is a primary key, or if it is to have an associated comment
- referential constraints on the table.

Generic clauses enable you to define extra column attributes for relational tables other than DB2 tables. The DB2 facilities FIELDPROC, EDITPROC, VALIDPROC, AUDIT, COMMENT and LABEL are supported by the FIELDPROC, EDITPROC, VALIDPROC, AUDIT, DB2-COMMENT and DB2-LABEL clauses respectively.

AS

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

IEW-DB2-TABLE

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

CREATOR-OWNER
To define the owner of an object, enter:

```
CREATOR-OWNER user
```

user is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the object.

On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member. The CREATOR-OWNER clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in an export to DB2 panel.

IN
To define the table space in which the table is created, enter:

```
IN tbspace
```

tbspace is the name of a IEW-TBSPACE member.

On encoding the member specified is checked to ensure it is a DB2-TBSPACE member.

For the successful generation of SQL statements the IEW-DB2-TABLE definition must include an IN clause, naming a IEW-TBSPACE member. The IEW-TBSPACE must include an IN clause, naming an IEW-DB2-DATABASE. This chain of relationships defines the database to which the table space, and table belong.

COLUMNS
To specify which form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns in the table, enter:

```
COLUMNS form-keyword
```

form-keyword is one of the following:

```
ENTERED-AS
HELD-AS
REPORTED-AS
DEFAULTED-AS
```

The form keyword that you define in the COLUMNS clause applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.

LIKE
In DB2, you may wish to create a table with the same column structure as an existing table, so that you can use it in production and development. To copy the columns of an existing table in DB2, enter:

```
LIKE member
```

member is the name of a DB2-TABLE, SQL-TABLE, DB2-VIEW or SQL-VIEW member.

EDITPROC
To define an edit routine for the table, enter:

```
EDITPROC process
```

process is the name of a SYSTEM, MMR-SYSTEM, PROGRAM or MODULE member.

The member represents an edit routine that must exist in DB2. In DB2, the edit routine is invoked whenever a row in the table is retrieved, updated or inserted.

IEW-DB2-TABLE

VALIDPROC

To define a validation routine for the table, enter:

```
VALIDPROC process
```

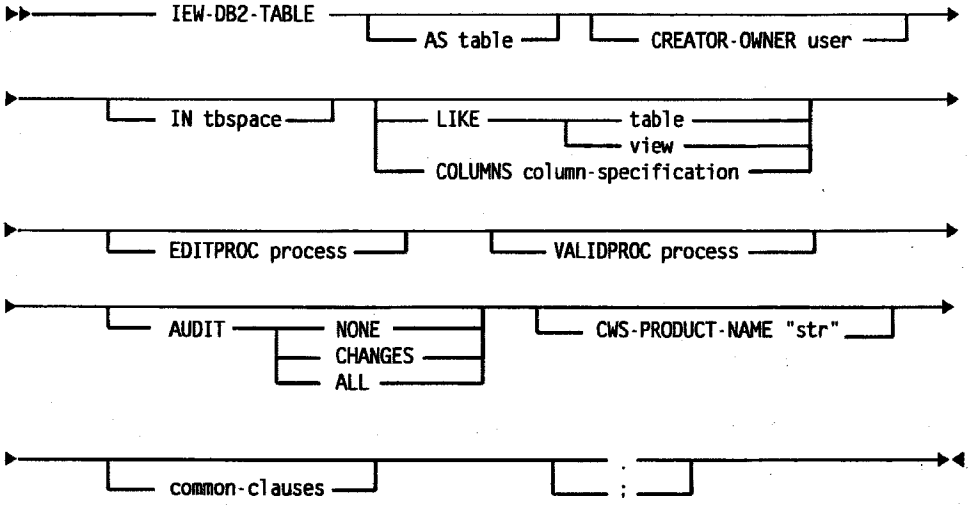
process is the name of a SYSTEM, MMRSYSTEM, PROGRAM or MODULE member.

The member represents a validation routine that must exist in DB2. In DB2, the validation routine receives an entire table row as input and may be used to control a subsequent INSERT, UPDATE or DELETE statement.

AUDIT

The auditing options for the table. Valid values are NONE, CHANGES and ALL.

Syntax



where

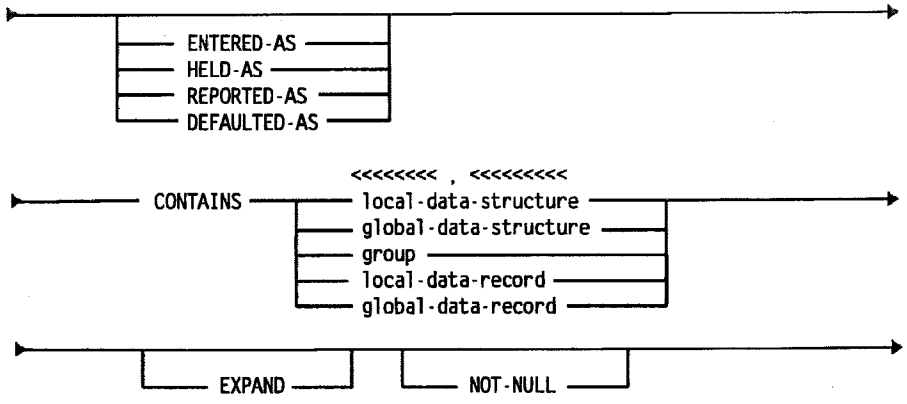
table is an IEW-DB2-TABLE member name

user is a DB2-USER member name

tbspace is an IEW-TBSPACE member name

view is an IEW-DB2-VIEW member name

column-specification is:



where

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

group is a GROUP member name

local-data-record is an IEW-LOCAL-DATA-RECORD member name

global-data-record is an IEW-GLOBAL-DATA-RECORD member name

process is the name of a SYSTEM, PROGRAM or MODULE member name

"str" is data to be imported from, or exported to, ADW/IEW

common clauses are any of the clauses common to all member types.

IEW-DB2-TABLE

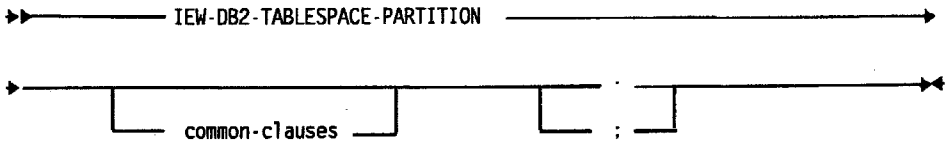
Refer to Appendix 2 for details of the common clauses.

IEW-DB2-TABLESPACE-PARTITION

ADW/IEW Design Workstation DB2-TABLESPACE-PARTITION member type.

Documentary object added to hold additional attributes other than those already held in the PARTITION clause of the related IEW-TBSPACE.

Syntax



where

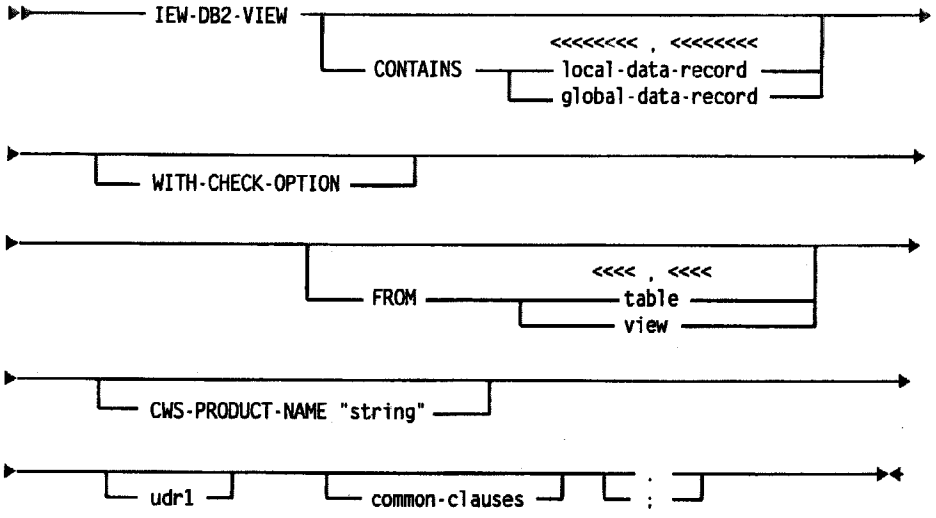
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-DB2-VIEW

ADW/IEW Design Workstation RELATIONAL-VIEW member type.

Syntax



where

local-data-record is an IEW-LOCAL-DATA-RECORD member name

global-data-record is an IEW-GLOBAL-DATA-RECORD member name

table is an IEW-DB2-TABLE member name

view is an IEW-DB2-VIEW member name

"string" is data to be imported from, or exported to, ADW/IEW

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-ENTITY-TYPE

IEW-ENTITY-TYPE

ADW/IEW Analysis Workstation ENTITY-TYPE member type. A class of people, places, things or concepts with characteristics of interest to the enterprise.

PURPOSE
Permitted values are:
- fundamental,
- associative,
- attributive,
- other,
- value,
- token or
- pointer

For a DATA-STRUCTURE:
Indicates whether the role of the data structure is to hold a VALUE, TOKEN, or POINTER.

For an ENTITY-TYPE:
The role that an entity type plays in describing the entity model.

Fundamental: Fundamental entity types are the most basic, free-standing entity types in the enterprise. The fundamental entity types are those that come most readily to mind as "people, places, things, or concepts of interest" - "Employee", "Department", "Customer", "Product" and the like.

Associative: Associative entity types exist primarily for representing the relationships between entity types. Associative entity types usually emerge when a known type of relationship between fundamental entity types turns out to have attribute types or relationship types of its own - e.g. "Employee works for DEPARTMENT" has an attribute type "appointment date".

Attributive: Attributive entity types exist mainly to describe fundamental or associative entity types. They are usually entity types that emerge when an attribute type of an established entity type turns out to have attribute types or relationship types of its own - e.g., the attribute type "EMPLOYEE.spouse" has an age of interest too.

CAUSES
Records the problems caused by this ENTITY-TYPE

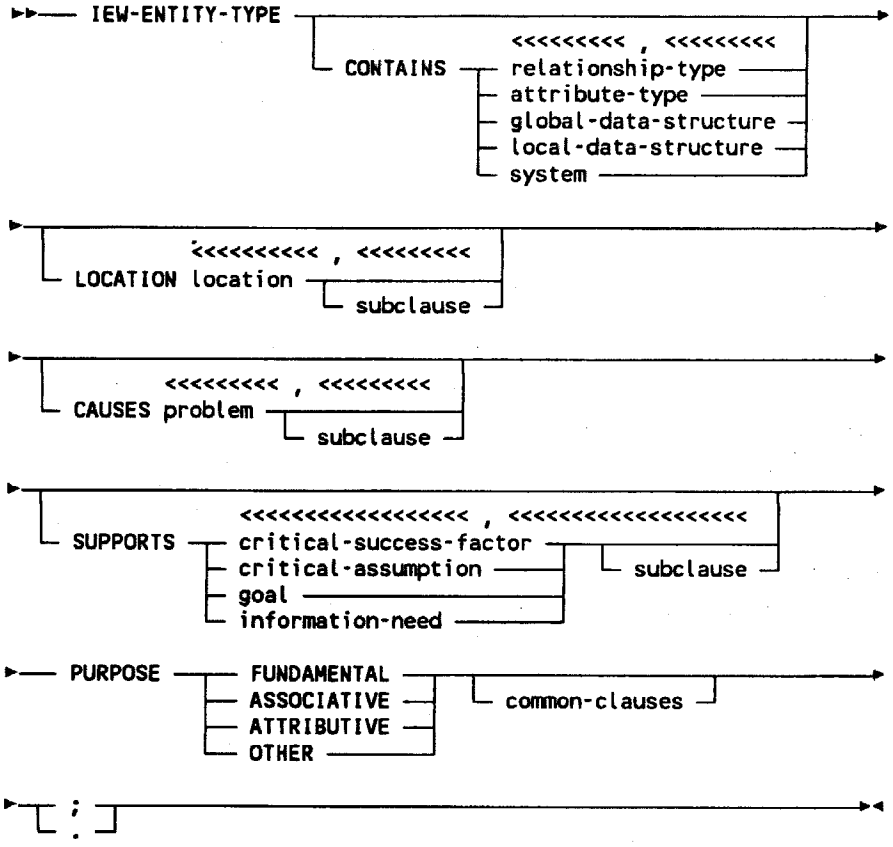
CONTAINS
Must contain the name of every data-structure which implements this ENTITY-TYPE.

SUPPORTS
Must contain the name of every planning object which is a part of this ENTITY-TYPE

LOCATION
States the location within the enterprise of this object by naming the appropriate LOCATION members.

IEW-ENTITY-TYPE

Syntax



where

relationship-type is an IEW-RELATIONSHIP-TYPE member name

attribute-type is an IEW-ATTRIBUTE-TYPE member name

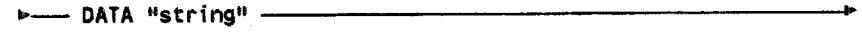
global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

system is a SYSTEM member name

location is an IEW-LOCATION member name

subclause is



where **"string"** is data to be imported from, or exported to, ADW/IEW

problem is an IEW-PROBLEM member name

critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member name

critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

goal is an IEW-GOAL member name

information-need is an IEW-INFORMATION-NEED member name

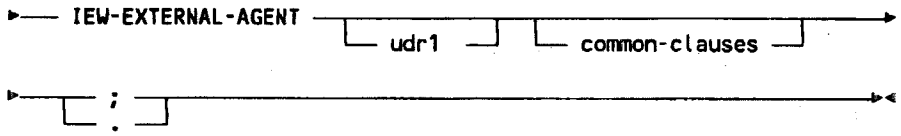
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-EXTERNAL-AGENT

ADW/IEW Analysis Workstation EXTERNAL-AGENT member type.
A process outside the scope of the enterprise model that is
capable of sending data to or receiving data from processes
within the model.

Syntax



where

udr1 is as defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-FIELD

ADW/IEW Design Workstation FIELD member type.

DEFAULTED-AS
This clause describes the form and/or values that the item defaults to
The clause is similar in form and content to the ENTERED-AS clause, except that only 1 version is permitted.
Example:

To express a default value

```
NUMERIC SIGNED
        CONTENTS IS 0
```

ENTERED-AS
specifies the form of the item when it is entered into the computer.
For full details of the power of this clause see the MSP publication "Dictionary/Repository Users Guide" (MPR-DRUG).
Examples:

To express the fact that the item is 30 characters long:

```
CHARACTER 30
```

To express the fact that the item has the same length and format as another item:

```
NAME IT-ZIPCODE
```

To express the fact that there are different versions of the item

```
  1  CHARACTER 30
ENTERED-AS                                  (ie:repeat the clause)
  2  CHARACTER 32
```

To express Cobol 88-level condition names:

```
NUMERIC UNSIGNED 2
      CONTENTS IS 25  CONDITION-NAME GREEN
          ELSE IS 15  CONDITION-NAME RED
          ELSE IS 05  CONDITION-NAME BLUE
```

To express the fact that the item can have a range of values:

```
NUMERIC SIGNED 2
      CONTENTS RANGE -30 TO +55, +70 TO +75
```

To express cross-dependencies on value in another item:

```
NUMERIC 2
      CONTENTS IS 25 IF IT-CREDIT-CODE = 3 OR IT-CREDIT-
                                                CODE = 5
          ELSE IS 30 IF IT-CREDIT-CODE = 4
```

To express the fact that the item is validated on entry by a module

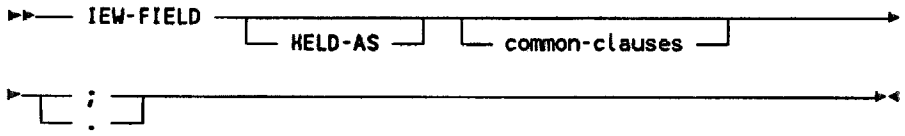
```
NUMERIC UNSIGNED 9
      USER-EXIT MO-CHECK-DIGIT
```

HELD-AS
This clause describes the form in which the item is held in the computer. The clause is similar in form and content to the ENTERED-AS clause.

REPORTED-AS
This clause describes the form in which the item is reported. The clause is similar in form and content to the ENTERED-AS clause.

IEW-FIELD

Syntax



where **common-clauses** are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

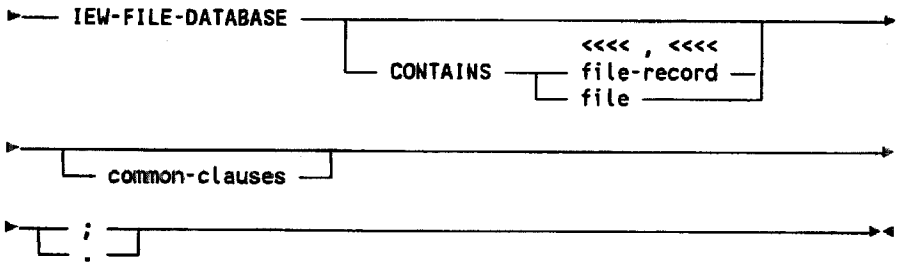
IEW-FILE-DATABASE

ADW/IEW Design Workstation MMT-FILE-DATABASE member type.

CONTAINS
Contains the names of one or more FILE-RECORDS
which compose the structure of this file.

IEW-FILE-DATABASE

Syntax



where

file-record is an IEW-FILE-RECORD member name

file is a FILE member name

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-FILE-RECORD

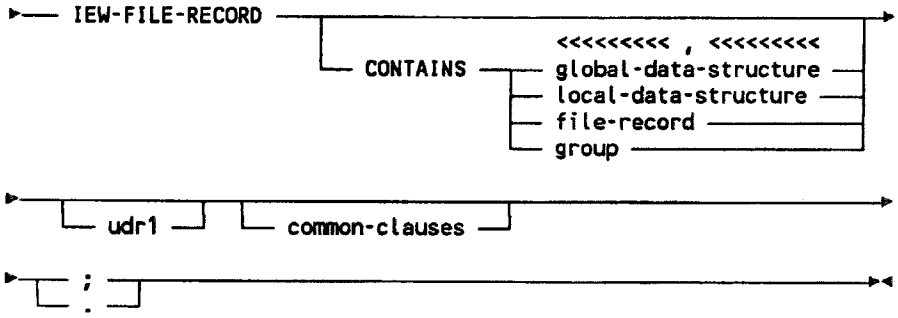
ADW/IEW Design Workstation FILE-RECORD member type.

CONTAINS
Must contain the name of every GLOBAL-DATA-STRUCTURE or LOCAL-DATA-STRUCTURE this file record is composed of.

SEE
List of LOCAL and GLOBAL-DATA-STRUCTURES describing this data-area.

IEW-FILE-RECORD

Syntax



where

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

file-record is an IEW-FILE-RECORD member name

group is a GROUP member name

udr1 is as defined in Appendix 2

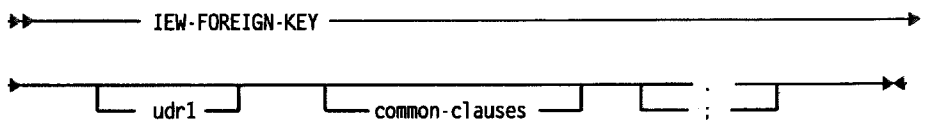
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-FOREIGN-KEY

ADW/IEW Design Workstation FOREIGN-KEY member type.

Syntax



where

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

IEW-FUNCTION

ADW/IEW Planning Workstation FUNCTION member type.
The group of processes that together support one aspect of furthering the mission of the enterprise.

HAS
This clause contains the names of the FUNCTIONS or PROCESSES that constitute this FUNCTION.

INFLUENCES
Specifies the SUBJECT-AREAS involved in this FUNCTION.

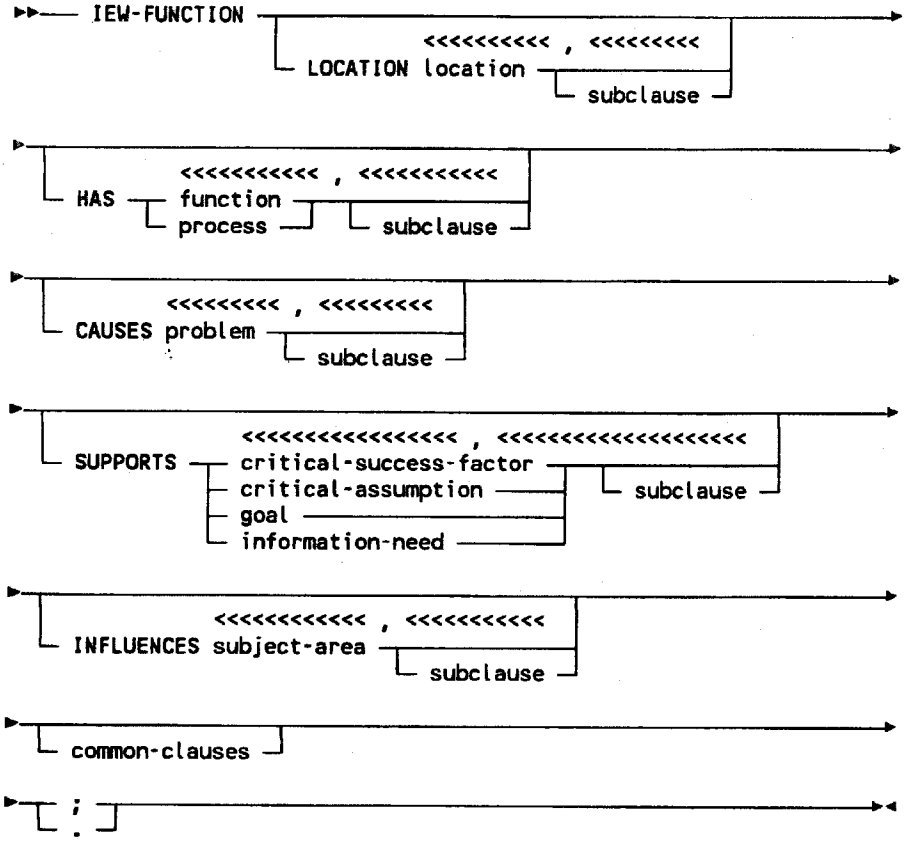
LOCATION
States the location within the enterprise of this object by naming the appropriate LOCATION members.

SUPPORTS
Must contain the name of every planning object which is a part of this FUNCTION.

CAUSES
Records the problems caused by this FUNCTION.

IEW-FUNCTION

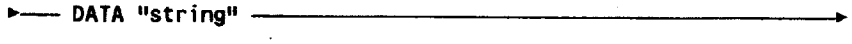
Syntax



where

location is an IEW-LOCATION member name

subclause is



where **"string"** is the data to be imported from, or exported to, ADW/IEW

function is an IEW-FUNCTION member name

process is an IEW-PROCESS member name

problem is an IEW-PROBLEM member name

critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member name

critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

goal is an IEW-GOAL member name

information-need is an IEW-INFORMATION-NEED member name

subject-area is an IEW-SUBJECT-AREA member name

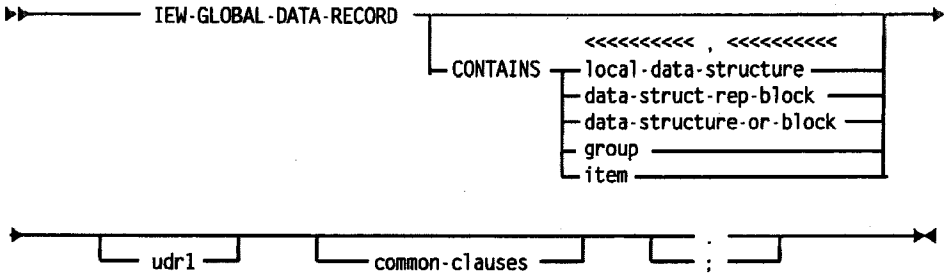
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-GLOBAL-DATA-RECORD

ADW/IEW Design Workstation GLOBAL-DATA-RECORD member type.

Syntax



where

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

data-struct-rep-block is an IEW-DATA-STRUCT-REP-BLOCK member name

data-structure-or-block is an IEW-DATA-STRUCTURE-OR-BLOCK member name

group is a GROUP member name

item is an ITEM member name

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

IEW-GLOBAL-DATA-STRUCTURE

ADW/IEW Design Workstation GLOBAL-DATA-STRUCTURE member type.

PURPOSE
Permitted values are:
- fundamental,
- associative,
- attributive,
- other,
- value,
- token or
- pointer

For a DATA-STRUCTURE:
Indicates whether the role of the data structure is to hold a VALUE, TOKEN, or POINTER.

For an ENTITY-TYPE:
The role that an entity type plays in describing the entity model.

Fundamental: Fundamental entity types are the most basic, free- standing entity types in the enterprise. The fundamental entity types are those that come most readily to mind as "people, places, things, or concepts of interest" - "Employee", "Department", "Customer", "Product" and the like.

Associative: Associative entity types exist primarily for representing the relationships between entity types. Associative entity types usually emerge when a known type of relationship between fundamental entity types turns out to have attribute types or relationship types of its own - e.g. "Employee works for DEPARTMENT" has an attribute type "appointment date".

Attributive: Attributive entity types exist mainly to describe fundamental or associative entity types. They are usually entity types that emerge when an attribute type of an established entity type turns out to have attribute types or relationship types of its own - e.g., the attribute type "EMPLOYEE.spouse" has an age of interest too.

HELD-AS
This clause is used if the member contains items and groups in the HELD-AS form. It is an alternative to ENTERED-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

ENTERED-AS
This clause is used if the member contains items and groups in the ENTERED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

REPORTED-AS
This clause is used if the member contains items and groups in the REPORTED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

DEFAULTED-AS
This clause is used if the member contains items and groups in the DEFAULTED-AS form. It is an alternative to HELD-AS, REPORTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

ALIGNED
This clause is used if all the members in the CONTAINS clause are ALIGNED. It is an alternative to UNALIGNED or NOT-ALIGNED. Only one such clause is valid in the definition.

UNALIGNED
This clause is used if all the members in the CONTAINS clause are UNALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

IEW-GLOBAL-DATA-STRUCTURE

NOT-ALIGNED
This clause is used if all the members in the CONTAINS clause are NOT-ALIGNED. It is an alternative to ALIGNED.
Only one such clause is valid in the definition.

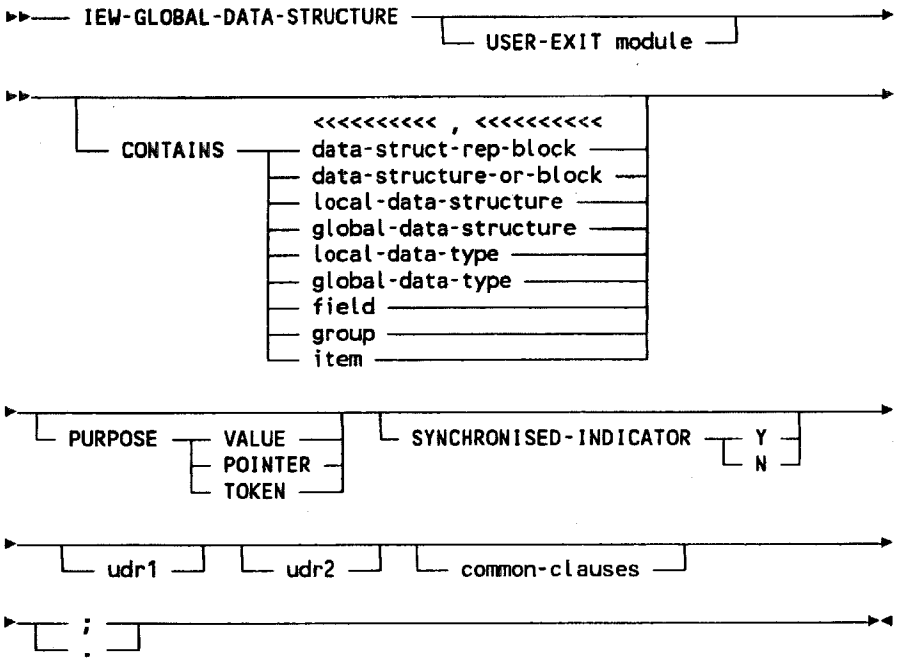
CONTAINS
Lists the names of DATA-STRUCTURE-OR-BLOCKS, DATA-STRUCTURE-REPETITION-BLOCKS, LOCAL-DATA-STRUCTURES or GLOBAL-DATA-STRUCTURES contained by this data-structure.

SEE
List of LOCAL and GLOBAL DATA-STRUCTURES tied to this data-structure.

USER-EXIT
Contains the name of a module by which this data structure is always processed, such as a validation module.

SYNCHRONISED-INDICATOR
Is this object synchronised, indicate Y or N.

Syntax



where

module is an IEW-MODULE member name

data-struct-rep-block is an IEW-DATA-STRUCT-REP-BLOCK member name

data-structure-or-block is an IEW-DATA-STRUCTURE-OR-BLOCK member name

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

global-data-type is an IEW-GLOBAL-DATA-TYPE member name

local-data-type is an IEW-LOCAL-DATA-TYPE member name

IEW-GLOBAL-DATA-STRUCTURE

field is an IEW-FIELD member name

group is a GROUP member name

item is an ITEM member name

udr1 and **udr2** are as defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-GLOBAL-DATA-TYPE

ADW/IEW Design Workstation GLOBAL-DATA-TYPE member type.

SQL-DATA-TYPE
SQL Data type

SQL-DATA-TYPE-MAX-LENGTH
Maximum length

SQL-DATA-TYPE-PRECISION
Precision

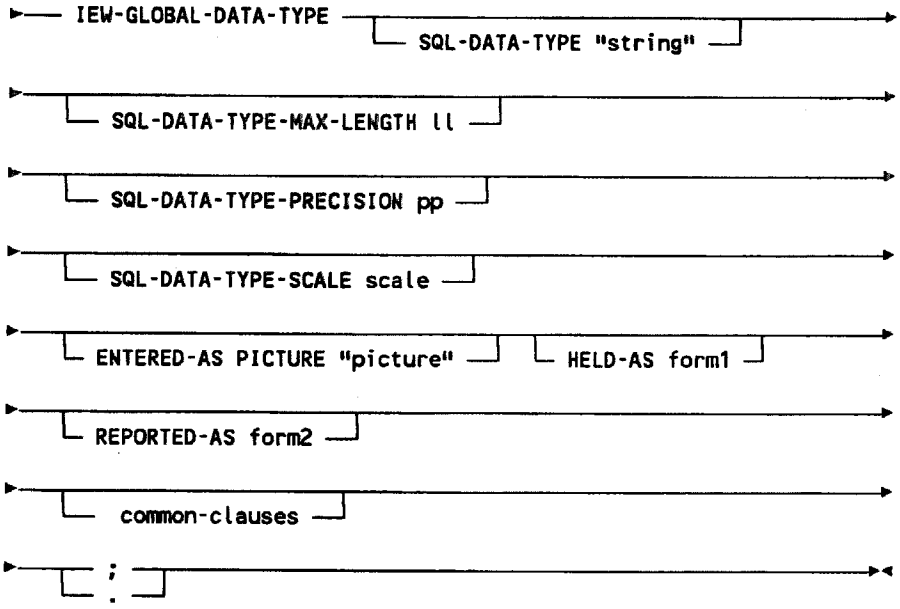
SQL-DATA-TYPE-SCALE
Scale

AS
The SQL/DS-specific definitions which are not explicitly given in this definition are taken from the SQL/DS member of the same type indicated here.

HELD-AS
This clause is used if the view contains items and groups in the HELD-AS form. It is an alternative to ENTERED-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition. If the view is ENTERED-AS, DEFAULTED-AS or REPORTED-AS add this clause immediately after the SQL-VIEW line and ignore the HELD-AS clause. If none of the above options is used the view is assumed to use the DEFAULTED-AS definitions of the component groups and items.

IEW-GLOBAL-DATA-TYPE

Syntax



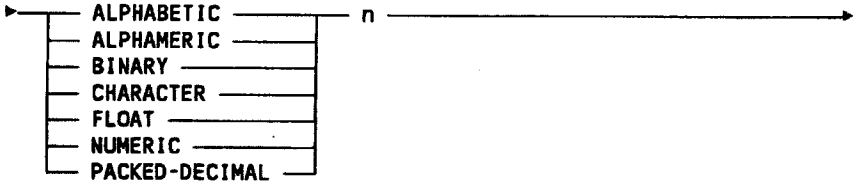
where

"string" is data for import from, or export to, ADW/IEW

ll, pp, and **scale** are unsigned integers

"picture" is a valid data-type format

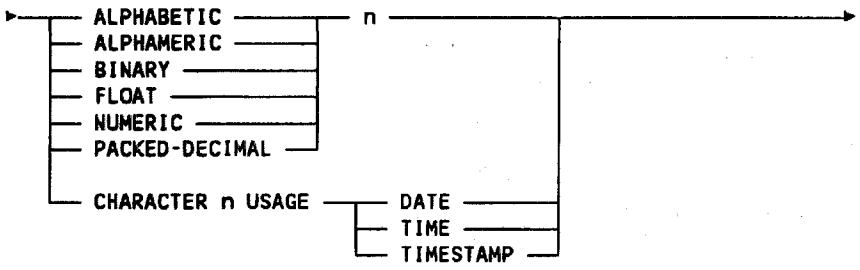
form1 is



where

n is an unsigned decimal number specifying a maximum length

form2 is



n is as defined above

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-GOAL

ADW/IEW Planning Workstation GOAL member type.
An end or target that all or part of the enterprise is committed to achieving (a mission or objective).

CAUSES
Records the problems caused by this GOAL.

HAS
This clause contains the names of the GOALS that constitute this GOAL.

RELIES-ON
Lists the GOALS that this GOAL itself requires.

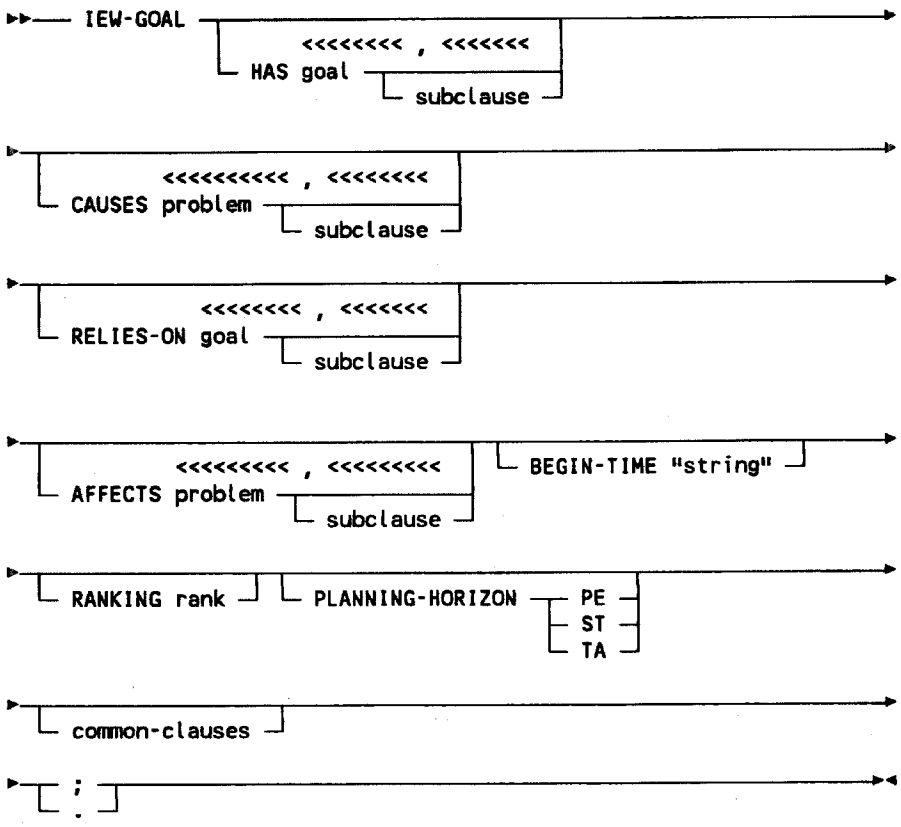
AFFECTS
Contains a list of the problems that this GOAL will solve.

BEGIN-TIME
Date from which planning entity is effective in the form MMM-YYYY.
The past, present or future date a critical assumption is initially effective.

RANKING
Relative importance of Planning Object.
An integer from 1-9999
How important is this critical assumption relative to other critical assumptions of the enterprise?

PLANNING-HORIZON
PE - Permanent (long term)
ST - Strategic (2 - 5 yrs)
TA - Tactical (0 - 2 yrs)

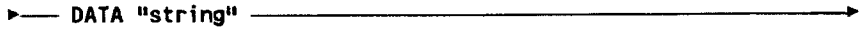
Syntax



where

goal is an IEW-GOAL member name

subclause is



where "**string**" is the data to be imported from, or exported to, ADW/IEW

problem is an IEW-PROBLEM member name

"**string**" is as defined above

IEW-GOAL

rank is an unsigned integer in the range 1 to 9999

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-INDEX

ADW/IEW Design Workstation RELATIONAL-INDEX member type.

UNIQUE

To define that indexed columns in the table being indexed do not have duplicate entries, enter:

```
UNIQUE
```

In DB2, when a single column in a table is being indexed, then each value can appear once only in the indexed column. When more than one column in a table is being indexed, then any given set of values can appear once only in the indexed columns.

ON

To define the table to which the index refers, enter:

```
ON table
```

table is the name of an IEW-DB2-TABLE or SQL-TABLE member. On encoding, the member specified in the ON clause is checked to ensure that it is a DB2-TABLE or SQL-TABLE member. The ON clause must be present for the successful generation of SQL CREATE INDEX statements.

STOGROUP

To define the VSAM catalog the index or partition is to use, enter:

```
VCAT catalog        for the partition
VCAT-1              for the index
```

catalog is the name of a VSAM catalog, of no more than 8 characters.

FREEPAGE

You can accommodate future expansion of an index, table space or partition by defining the frequency with which pages are left free.

To define the relative frequency with which free pages are allocated, enter:

```
FREEPAGE-1 fn        for the index
FREEPAGE fn          for the partition
```

fn is an integer in the range 0 to 255.

For example, FREEPAGE 4 means that 1 free page is left after every 4 pages.

If you do not define a FREEPAGE clause the DB2 defaults apply.

PCTFREE
You can accommodate future expansion of an index, table space or partition by defining the percentage of each page that is left free.

To define the percentage space kept free on a page, when an index, table space or partition is loaded or reorganized, enter:

```
PCTFREE-1 pn         for the index
PCTFREE pn           for the partition
```

pn is an integer in the range 0 to 99 .

If you do not define a PCTFREE clause the DB2 defaults apply.

CLUSTER
To define a clustered index, enter:

```
CLUSTER
```

Partitioned indexes must also be clustered. If you do not include the CLUSTER keyword in a DB2-INDEX that contains a PARTITION clause, it is automatically generated in SQL CREATE INDEX statements. A warning message is also generated, to remind you to include the CLUSTER keyword in the member definition.

PARTITION
To define that an index or table space is to be partitioned, enter:

```
PARTITION
```

You can optionally define a number, details of storage, and free space allocation for each partition.

To give the partition a number, enter:

```
NUMBER n
```

n is an integer in the range 1 to 64.

To define the storage space to which the partition belongs, use the VCAT, or PRIQTY, SECQTY and ERASE clauses. To define how much space is left free in the partition, use the FREEPAGE and PCTFREE clauses.

If you do not define storage or free space for partitions individually, they automatically take space from that specified for the index or table space as a whole.

SUBPAGES
Each physical page of an index may be divided into 1, 2, 4, 8 or 16 subpages. Each subpage is a unit of locking.

To define subpages and therefore the locking unit, enter:

```
SUBPAGES division
```

division is one of the following integers:

1
2
4
8
16

If you do not define a locking parameter, none is generated, and the DB2 default applies.

BUFFERPOOL
To define the buffer pool that indexes use, enter:

```
BUFFERPOOL bufferpool-name
```

bufferpool-name is one of the following buffer pools:

```
BP0             BP1
BP2             BP32K
```

The buffer pool defined in the DB2-DATABASE definition is the default used by those indexes that belong to the database, and do not have a buffer pool specified in their own member definition.

CLOSE
To define that the data set, on which an index or table space resides, is to be closed when not in use, enter:

```
CLOSE YES
```

To define that it is to remain open, enter:

```
CLOSE NO
```

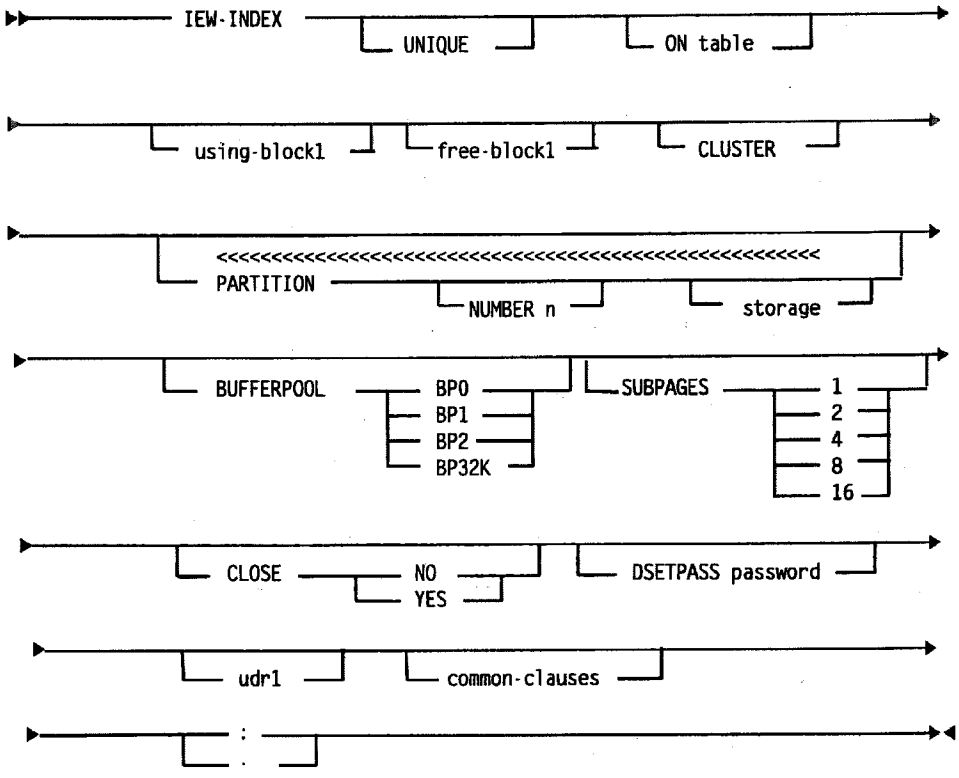
If you do not define a CLOSE clause the DB2 default applies.

DSETPASS
To define a password for the VSAM data set, on which an index or table space resides, enter:

```
DSETPASS password
```

password is a VSAM data set password of no more than 8 characters.

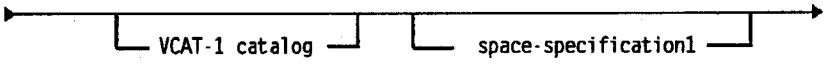
Syntax



where

table is an IEW-DB2-TABLE member name

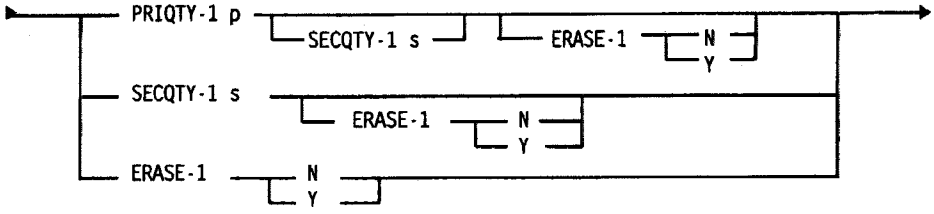
using-block1 is



where

catalog is a VSAM catalog name, of no more than 8 characters

space-specification1 is

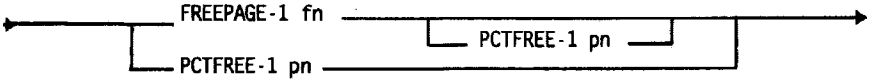


where

p is an integer in the range 3 to 4194304

s is an integer in the range 0 to 131068

free-block1 is



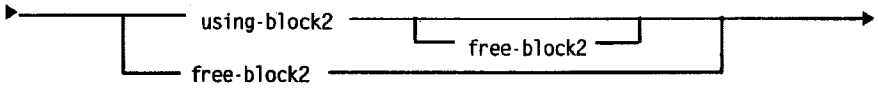
where

fn is an integer in the range 0 to 255

pn is an integer in the range 0 to 99

n is an integer in the range 1 to 64

storage is



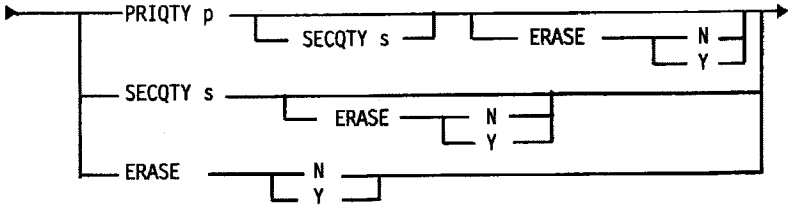
where

using-block2 is

where

catalog is as defined above

space-specification2 is

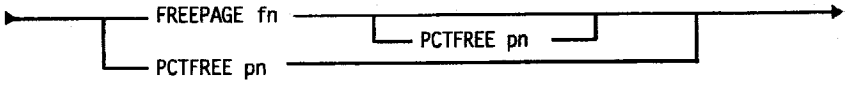


where

p is as defined above

s is as defined above

free-block2 is



where

fn is as defined above

pn is as defined above

password is a VSAM data set password, of no more than 8 characters

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

IEW-INFORMATION-NEED

IEW-INFORMATION-NEED

ADW/IEW Planning Workstation INFORMATION-NEED member type. A statement of need for information that is critical to the success of the enterprises goals.

HAS
This clause contains the names of the INFORMATION-NEEDS that constitute this INFORMATION-NEED.

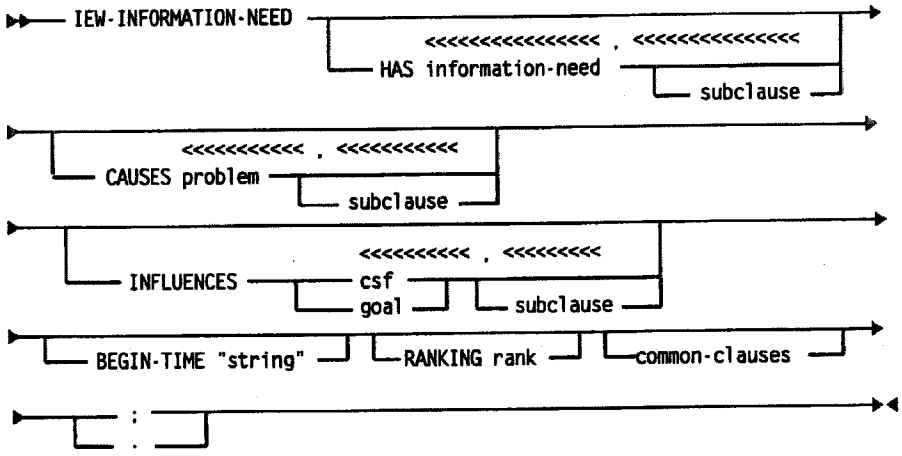
INFLUENCES
Specifies the CSF's and GOALS impacted by this INFORMATION-NEED.

CAUSES
Records the problems caused by this INFORMATION-NEED.

BEGIN-TIME
Date from which planning entity is effective in the form MMM-YYYY.
The past, present or future date a critical assumption is initially effective.

RANKING
Relative importance of Planning Object.
An integer from 1-9999
How important is this critical assumption relative to other critical assumptions of the enterprise?

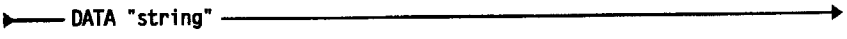
Syntax



where

information-need is an IEW-INFORMATION-NEED member name

subclause is



where **"string"** is the data to be imported from, or exported to, ADW/IEW

problem is an IEW-PROBLEM member name

csf is an IEW-CRITICAL-SUCCESS-FACTOR member name

goal is an IEW-GOAL member name

"string" is as defined above

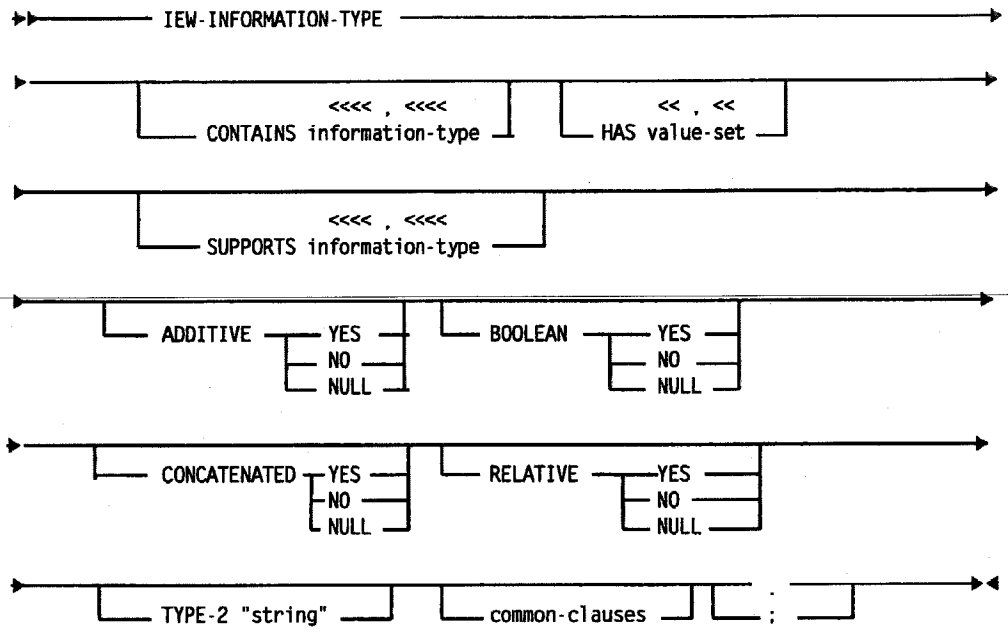
rank is an unsigned integer in the range 1 to 9999

common-clauses are any of the clauses available to all member types. Refer to Appendix 2 for details of the common clauses.

IEW-INFORMATION-TYPE

ADW/IEW Analysis Workstation INFORMATION-TYPE member type.

Syntax



where

information-type is an IEW-INFORMATION-TYPE member name

value-set is an IEW-VALUE-SET member name

"string" is data to be imported from, or exported to, ADW/IEW

common-clauses are any of the clauses available to all member types

Refer to Appendix 2 for details of the common clauses.

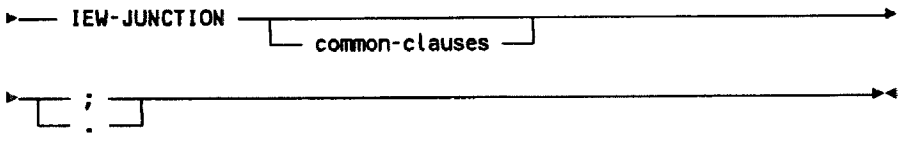
This page is intentionally blank.

IEW-JUNCTION

IEW-JUNCTION

ADW/IEW Analysis Workstation JUNCTION.
A point at which two or more dataflows are directly connected.
Junctions are unnamed.

Syntax



where **common-clauses** are any of the clauses available to all member types.

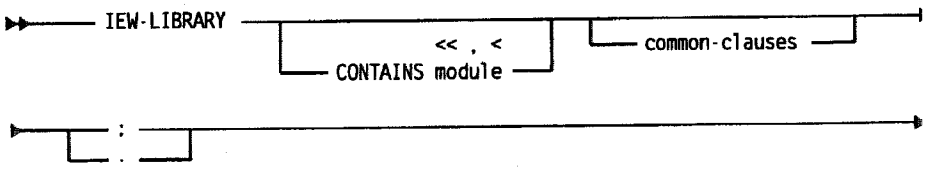
Refer to Appendix 2 for details of the common clauses.

IEW-LIBRARY

ADW/IEW Design Workstation LIBRARY member type.

CONTAINS
Must contain the name of every MODULE which is a part of this LIBRARY.

Syntax



where

module is an IEW-MODULE member name

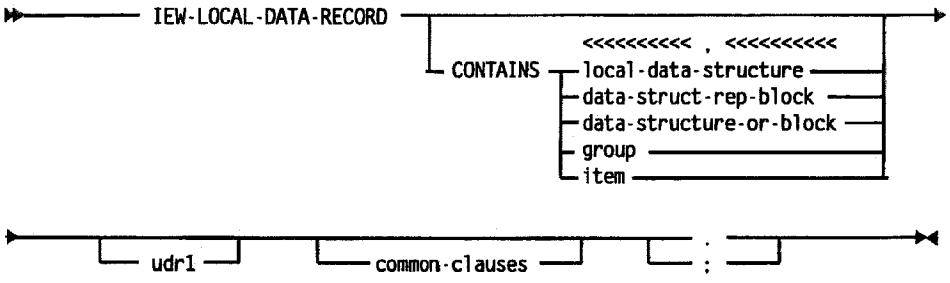
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-LOCAL-DATA-RECORD

ADW/IEW Design Workstation LOCAL-DATA-RECORD member type.

Syntax



where

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

data-struct-rep-block is an IEW-DATA-STRUCT-REP-BLOCK member name

data-structure-or-block is an IEW-DATA-STRUCTURE-OR-BLOCK member name

group is a GROUP member name

item is an ITEM member name

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

IEW-LOCAL-DATA-STRUCTURE

ADW/IEW Design Workstation LOCAL-DATA-STRUCTURE member type.

PURPOSE
Permitted values are:

fundamental,
associative,
attributive,
other,
value,
token or
pointer

For a DATA-STRUCTURE:
Indicates whether the role of the data structure is to hold a VALUE, TOKEN, or POINTER.

For an ENTITY-TYPE:
The role that an entity type plays in describing the entity model.

Fundamental: Fundamental entity types are the most basic, free-standing entity types in the enterprise. The fundamental entity types are those that come most readily to mind as "people, places, things, or concepts of interest" - "Employee", "Department", "Customer", "Product" and the like.

Associative: Associative entity types exist primarily for representing the relationships between entity types. Associative entity types usually emerge when a known type of relationship between fundamental entity types turns out to have attribute types or relationship types of its own - e.g. "Employee works for DEPARTMENT" has an attribute type "appointment date".

Attributive: Attributive entity types exist mainly to describe fundamental or associative entity types. They are usually entity types that emerge when an attribute type of an established entity type turns out to have attribute types or relationship types of its own - e.g., the attribute type "EMPLOYEE.spouse" has an age of interest too.

HELD-AS
This clause is used if the member contains items and groups in the HELD-AS form. It is an alternative to ENTERED-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

ENTERED-AS
This clause is used if the member contains items and groups in the ENTERED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition.

REPORTED-AS
This clause is used if the member contains items and groups in the REPORTED-AS form. It is an alternative to HELD-AS, DEFAULTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

DEFAULTED-AS
This clause is used if the member contains items and groups in the DEFAULTED-AS form. It is an alternative to HELD-AS, REPORTED-AS and ENTERED-AS. Only one such clause is valid in the definition.

ALIGNED
This clause is used if all the members in the CONTAINS clause are ALIGNED. It is an alternative to UNALIGNED or NOT-ALIGNED. Only one such clause is valid in the definition.

UNALIGNED
This clause is used if all the members in the CONTAINS clause are UNALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

IEW-LOCAL-DATA-STRUCTURE

NOT-ALIGNED
This clause is used if all the members in the CONTAINS clause are NOT-ALIGNED. It is an alternative to ALIGNED. Only one such clause is valid in the definition.

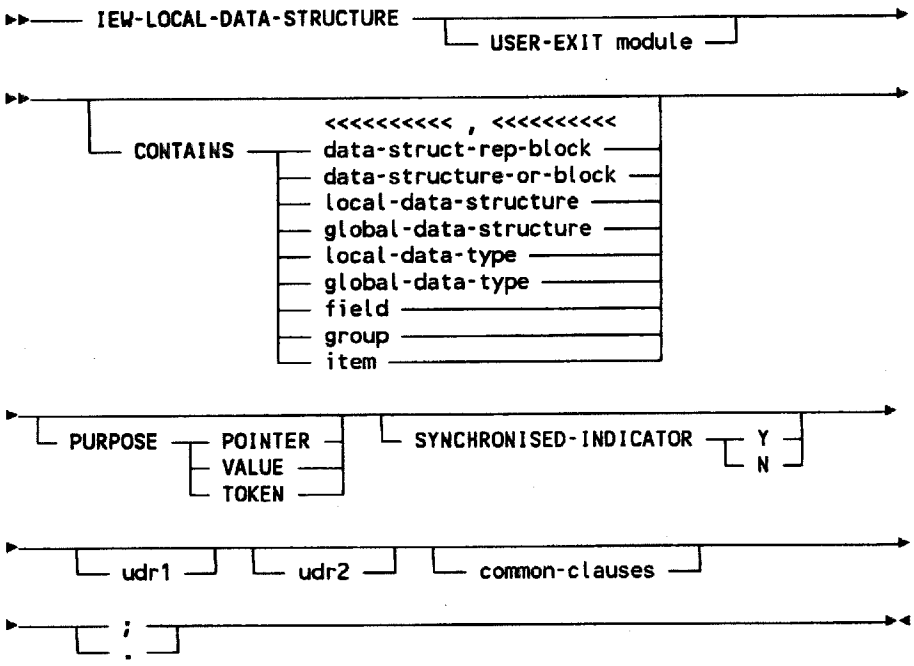
CONTAINS
Lists the names of DATA-STRUCTURE-OR-BLOCKS, DATA-STRUCTURE-REPETITION-BLOCKS, LOCAL-DATA-STRUCTURES or GLOBAL-DATA-STRUCTURES contained by this data-structure.

SEE
List of LOCAL and GLOBAL DATA-STRUCTURES tied to this data-structure.

USER-EXIT
Contains the name of a module by which this data structure is always processed, such as a validation module.

SYNCHRONISED-INDICATOR
Is this object synchronised, indicate Y or N

Syntax



where

module is an IEW-MODULE member name

data-struct-rep-block is an IEW-DATA-STRUCT-REP-BLOCK member name

data-structure-or-block is an IEW-DATA-STRUCTURE-OR-BLOCK member name

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

global-data-type is an IEW-GLOBAL-DATA-TYPE member name

local-data-type is an IEW-LOCAL-DATA-TYPE member name

IEW-LOCAL-DATA-STRUCTURE

field is a IEW-FIELD member name

group is a GROUP member name

item is an ITEM member name

udr1 and **udr2** are as defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-LOCAL-DATA-TYPE

ADW/IEW Design Workstation LOCAL-DATA-TYPE member type.

AS
The SQL/DS-specific definitions which are not explicitly given in this definition are taken from the SQL/DS member of the same type indicated here.

HELD-AS
This clause is used if the view contains items and groups in the HELD-AS form. It is an alternative to ENTERED-AS, DEFAULTED-AS and REPORTED-AS. Only one such clause is valid in the definition. If the view is ENTERED-AS, DEFAULTED-AS or REPORTED-AS add this clause immediately after the SQL-VIEW line and ignore the HELD-AS clause. If none of the above options is used the view is assumed to use the DEFAULTED-AS definitions of the component groups and items.

SQL-DATA-TYPE
SQL Data type

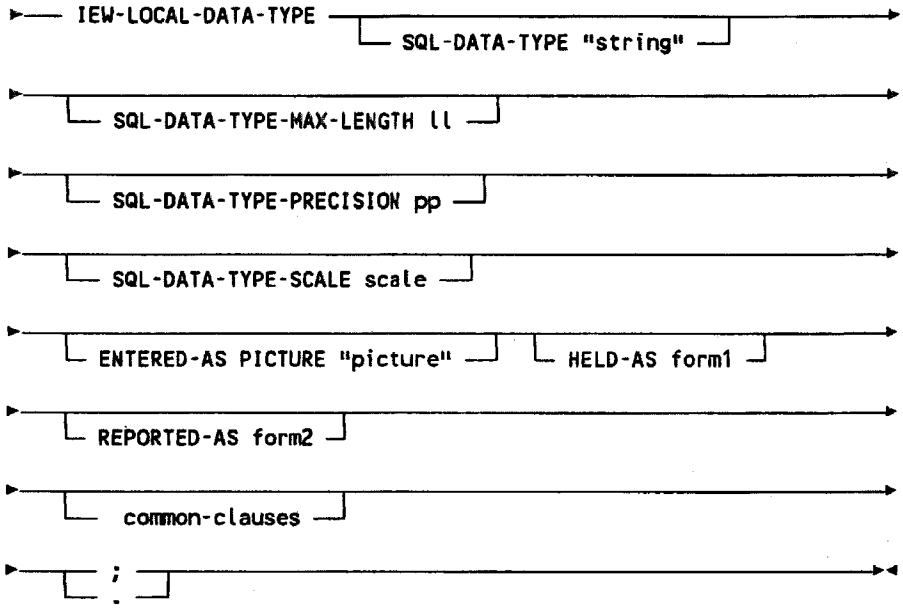
SQL-DATA-TYPE-MAX-LENGTH
Maximum length

SQL-DATA-TYPE-PRECISION
Precision

SQL-DATA-TYPE-SCALE
Scale

IEW-LOCAL-DATA-TYPE

Syntax



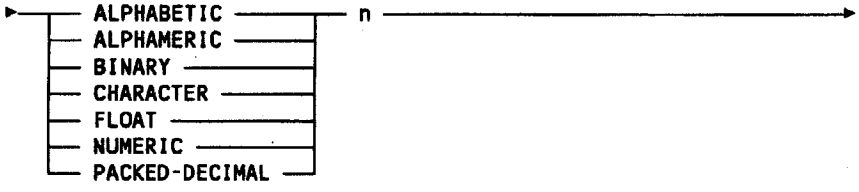
where

"string" is data for import from, or export to, ADW/IEW

ll, pp, and **scale** are unsigned integers

"picture" is a valid data-type format

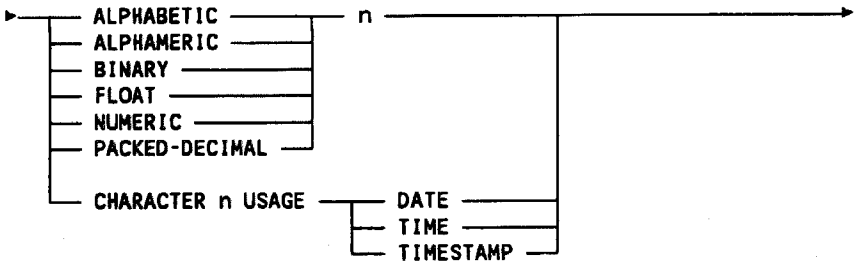
form1 is



where

n is an unsigned decimal number specifying a maximum length

form2 is



n is as defined above

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-LOCATION

ADW/IEW Planning Workstation LOCATION member type.
A physical place in the enterprise where processes are performed and/or data are recorded and maintained.

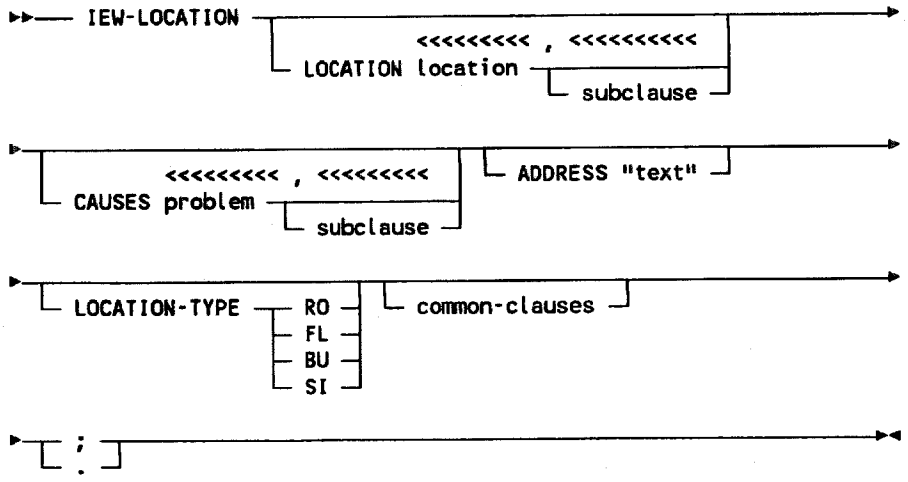
CAUSES
Records the problems caused by this LOCATION.

LOCATION
What other locations does this location consist of?

ADDRESS
Up to five lines of text.
This is a textual description of where the location is. Typically, such a description might consist of a full mailing address, or simply a specification of a city or a region.

LOCATION-TYPE
Code indicating the area this location represents:
RO - ROOM
FL - FLOOR
BU - BUILDING
SI - SITE

Syntax



where

location is an IEW-LOCATION member name

problem is an IEW-PROBLEM member name

"text " is up to 32767 delimited strings, each string having a maximum length of 60 characters

subclause is

where **"string"** is the data to be imported from, or exported to, ADW/IEW

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-MECHANISM

ADW/IEW Planning Workstation MECHANISM member type.
A method by which specific processes are executed, for example, a computer application.

CAUSES
Records the problems and mechanisms caused by this MECHANISM.

HAS
This clause contains the names of the MECHANISMS that constitute this MECHANISM.

IMPLEMENTS
Contains the name of each FUNCTION or PROCESS implemented implemented by this MECHANISM.

SUPPORTS
Must contain the name of every planning object which is a part of this MECHANISM.

AFFECTS
Records the mechanisms preceded by this MECHANISM.

CALLS
Records the data-collections called by this MECHANISM.

COMPLETENESS-RATING
Completeness of implementation
Values H(igh) M(edium) L(ow) or integer from 1 - 999

DATE-LAST-MODIFIED
Date in the form: mm/dd/yyyy
The date of the most recent major design modification of the Data Collection.

DISPOSITION
Indication of what to do with an existing Planning Object
A character code that indicates what planners have decided to do with an existing data collection.

KE - keep as is
IN - interface, or
RE - replace

KEep as is: The data collection is an acceptable implementation of its associated entity types. Retain it substantially unchanged.

INterface: The data collection needs work to be able to interface with the rest of the information system.

REplace: The data collection should be replaced with a new one.

IN - Interface with rest of system

RE - Replace

DISPOSITION-RATIONALE
Text supporting the decision on DISPOSITION
Up to five lines of text.
A textual explanation supporting the planners' decision of what to do with an existing object. Especially in cases where a decision as to what to do with an object might be questioned, this property will allow planners to document the reasons why they made that decision.

DOCUMENTATION-RATING
How well documented is this Planning Object?
Values H(igh) M(edium) L(ow) or integer from 1 - 999

EFFICIENCY-RATING
Rating of efficiency in terms of performance.
Values H(igh) M(edium) L(ow) or integer from 1 - 999

FLEXIBILITY-RATING
Rating in terms of compatibility, extensibility and portability

Values H(igh) M(edium) L(ow) or integer from 1 - 999

IEW-MECHANISM

MAINTAINABILITY-RATING
Rating in terms of simplicity, expandibility and portability.
Values H(igh) M(edium) L(ow) or integer from 1 - 999

RELIABILITY-RATING
Reliability under normal and fault conditions
Values H(igh) M(edium) L(ow) or integer from 1 - 999

RESPONSE-TIME
IM - Immediate (3 secs or less)
IN - Interactive (1 minute or less)
SA - Same day
OV - Overnight

STATUS
Code used to describe the state of the members types.
The codes have the following meanings:
PL - PLANNED
CU - CURRENT
PA - PAST
UD - UNDER DEVELOPMENT
AA - CREATED BY AFFINITY ANALYSIS

TECHNOLOGY

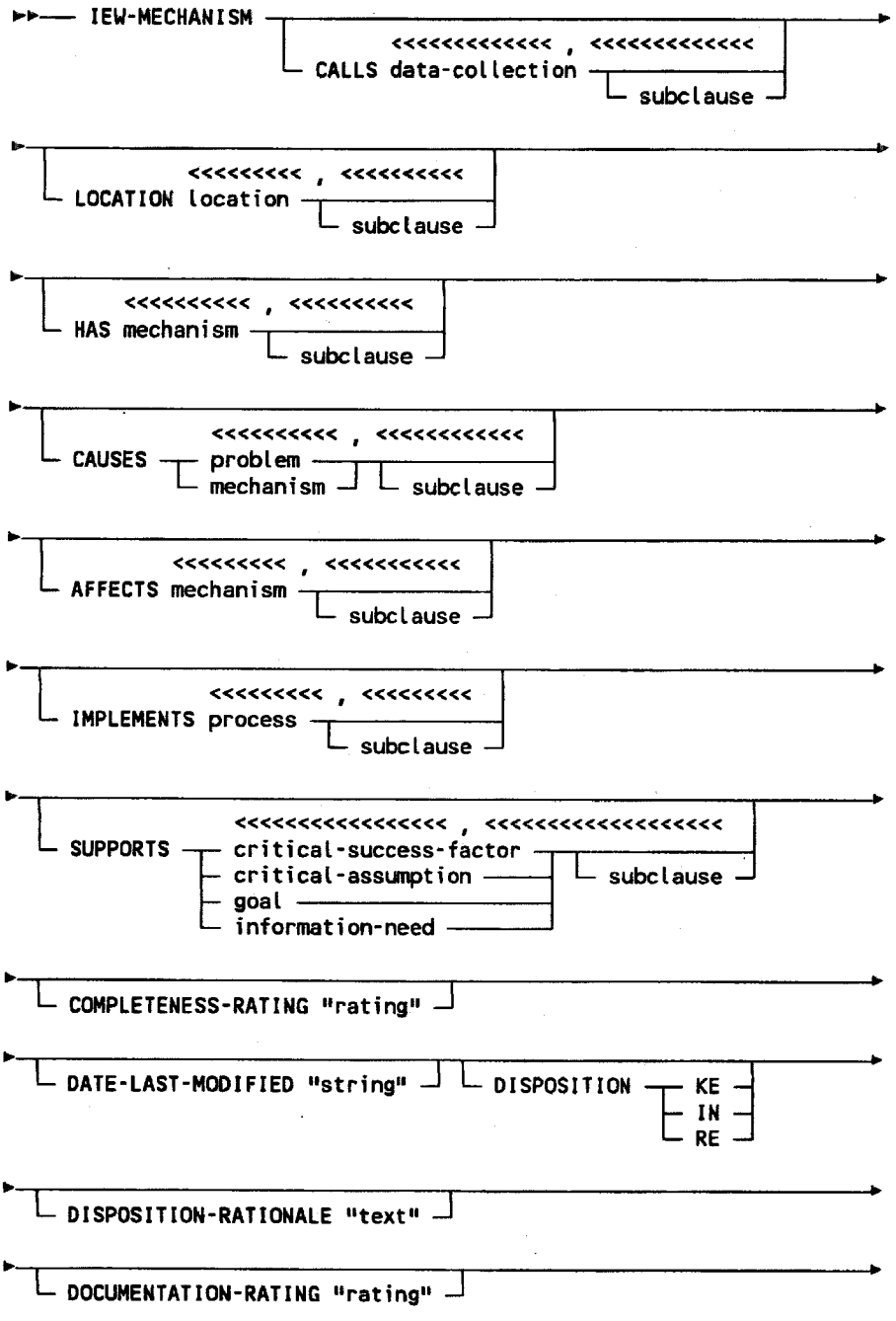
Data Collections	Mechanisms
HI - Hierarchical DBMS	AU - Automated
RE - Relational DBMS	MA - Manual
NT - Network DBMS	BO - Both
DB - DBMS maintained	
FI - Computer files	
MF - Microfiche files	
PA - Paper files	

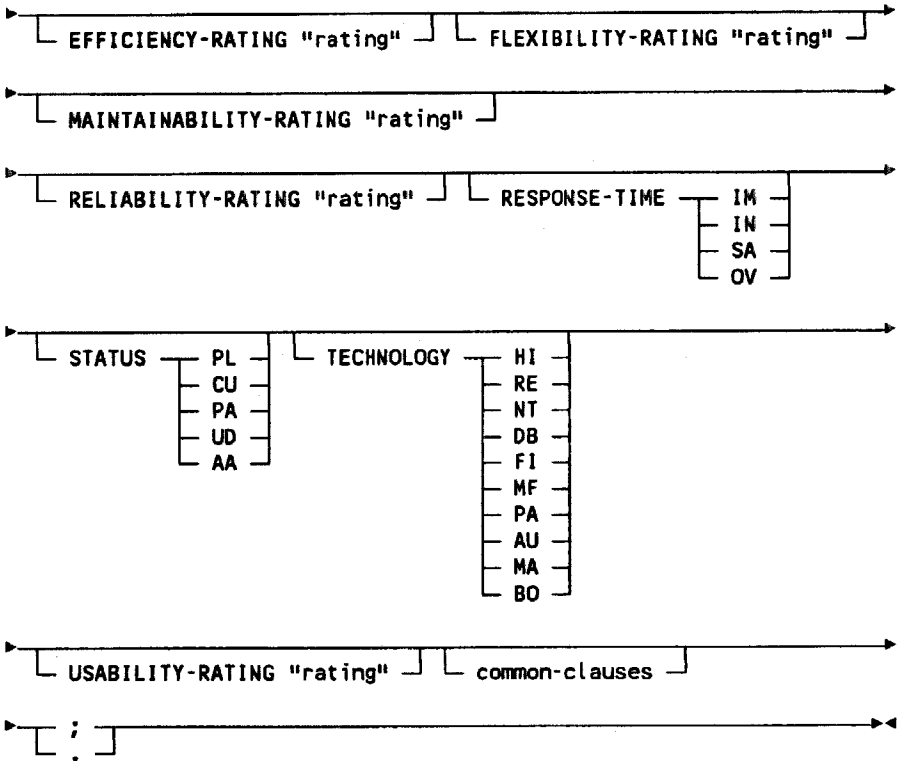
USABILITY-RATING
Reliability under normal and fault conditions
Values H(igh) M(edium) L(ow) or integer from 1 - 999

LOCATION
States the location within the enterprise of this object by naming the appropriate LOCATION members.

IEW-MECHANISM

Syntax

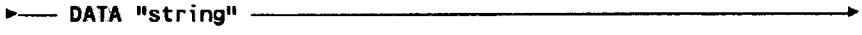




where

data-collection is an IEW-DATA-COLLECTION member name

subclause is



where

"string" is data to be imported from, or exported to, ADW/IEW

location is an IEW-LOCATION member name

mechanism is an IEW-MECHANISM member name

problem is an IEW-PROBLEM member name

IEW-MECHANISM

process is an IEW-PROCESS member name

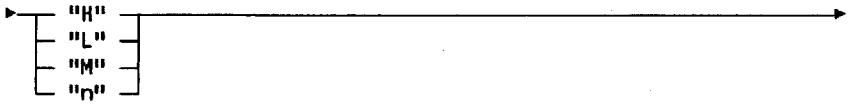
critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member name

critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

goal is an IEW-GOAL member name

information-need is an IEW-INFORMATION-NEED member name

"**rating**" is



"**n**" is an integer in the range "1" to "999"

"**string**" is as defined above

"**text** " is up to 32767 delimited strings, each string having a maximum length of 60 characters

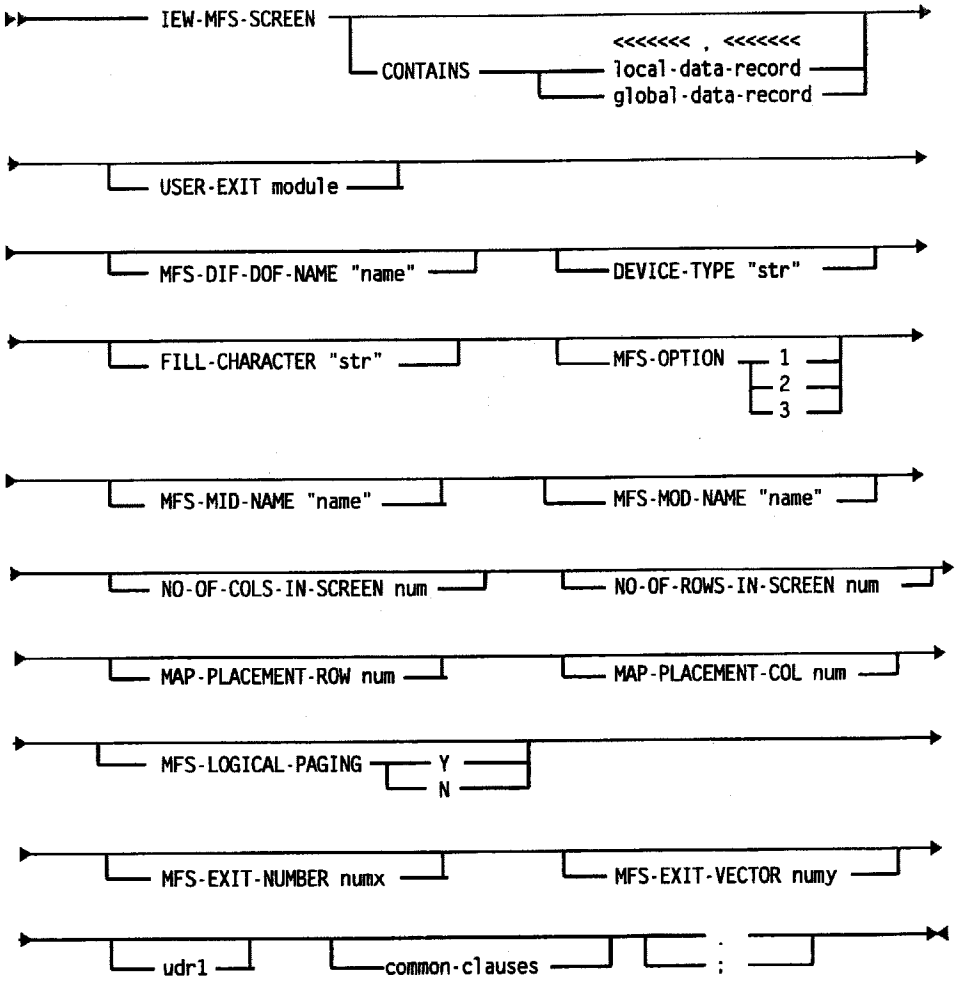
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-MFS-SCREEN

ADW/IEW Design Workstation MFS-SCREEN member type.

Syntax



where

local-data-record is an IEW-LOCAL-DATA-RECORD member name

IEW-MFS-SCREEN

global-data-record is an IEW-GLOBAL-DATA-RECORD member name

module is an IEW-MODULE member name

num is a positive integer

numx is a positive integer between 0 and 127

numy is a positive integer between 0 and 255

"**name**" is data to be imported from, or exported to, ADW/IEW

"**string**" is data to be imported from, or exported to, ADW/IEW

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-MODELLING-SOURCE

ADW/IEW Planning Workstation MODELLING-SOURCE member type.
A person, group or document that provides the information
that is used to model some aspect of the enterprise.

IMPLEMENTS
A list of planning objects that this MODELLING-SOURCE is the information source of.

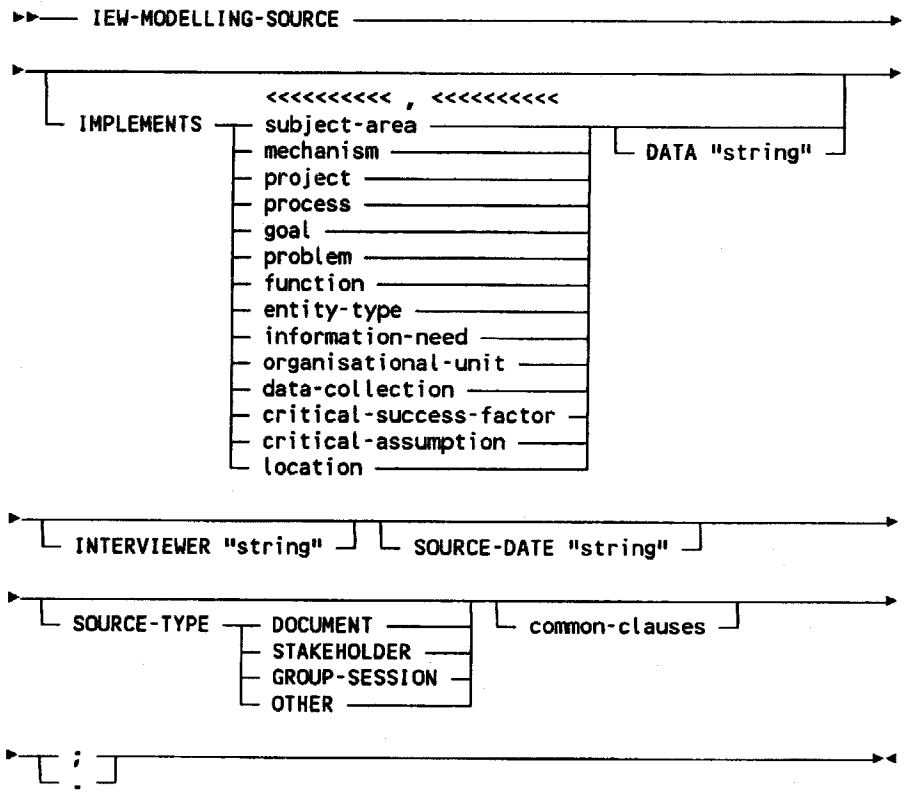
INTERVIEWER
1-32 characters
The name of the planner who reviews the document, conducts the interview, or moderates the group session.

SOURCE-DATE
Date of access
IEW format MM/DD/YYYY

SOURCE-TYPE
Indicates the type of modelling source
Possible values are:
DOCUMENT,
STAKEHOLDER,
GROUP-SESSION,
or OTHER.

IEW-MODELLING-SOURCE

Syntax



where

subject-area is an IEW-SUBJECT-AREA member name

mechanism is an IEW-MECHANISM member name

project is an IEW-PROJECT member name

process is an IEW-PROCESS member name

goal is an IEW-GOAL member name

problem is an IEW-PROBLEM member name

function is an IEW-FUNCTION member name

entity-type is an IEW-ENTITY-TYPE member name

information-need is an IEW-INFORMATION-NEED member name

organisational-unit is an IEW-ORGANISATIONAL-UNIT member name

data-collection is an IEW-DATA-COLLECTION member name

critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member

critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

location is an IEW-LOCATION member name

"string" is the data to be imported from, or exported to, ADW/IEW

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-MODULE

ADW/IEW Design Workstation MODULE member type.

CALLS
This clause may contain the name of one or more MODULE members. It may also contain one AT subclause, naming the point at which the module is called, and a PASSING clause naming the data members passed to it. Only one AT subclause can be specified in a CALLS clause. To define multiple called modules, each with its own AT subclause, multiple CALLS clauses must be specified.

SEE
List of SEQUENTIAL-PROCESSES and MODULES enacted by this module.

INPUTS
Contains the names of any SCREENS, FILE-RECORDS, RELATIONS, or SEGMENTS that this module gets

PARAMETERS
Contains the names of PARAMETER members which define parameters passed to this member. It may also include an ENTRY-POINT subclause for each data-name, containing the label of an entry point in this member.

CONTAINS
Contains the names of any MODULE members used by this module.

OUTPUTS
Lists the names of SCREENS that this module outputs.

PREDEFINED
Is this module predefined Y or N

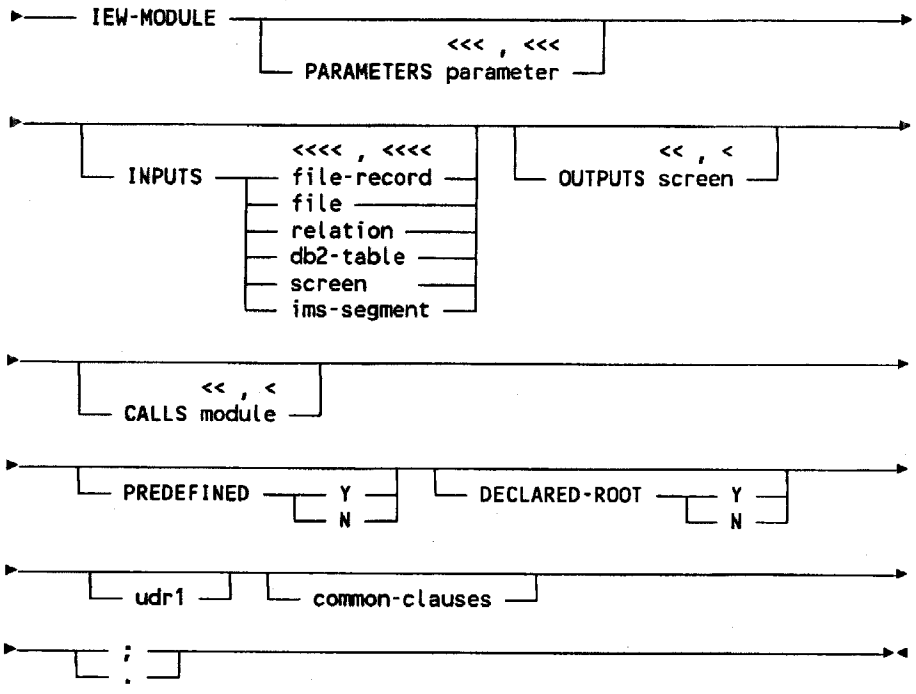
DECLARED-ROOT
Y - yes
N - no

Declaring an object a root states your intention of viewing the object as a separate decomposition diagram.

It does not automatically create a separate decomposition diagram.

IEW-MODULE

Syntax



where

parameter is an IEW-PARAMETER member name

file-record is an IEW-FILE-RECORD member name

file is a FILE member name

relation is an IEW-RELATION member name

db2-table is a DB2-TABLE member name

screen is an IEW-SCREEN member name

ims-segment is a SEGMENT member name

module is an IEW-MODULE member name

udr1 is as defined in Appendix 2

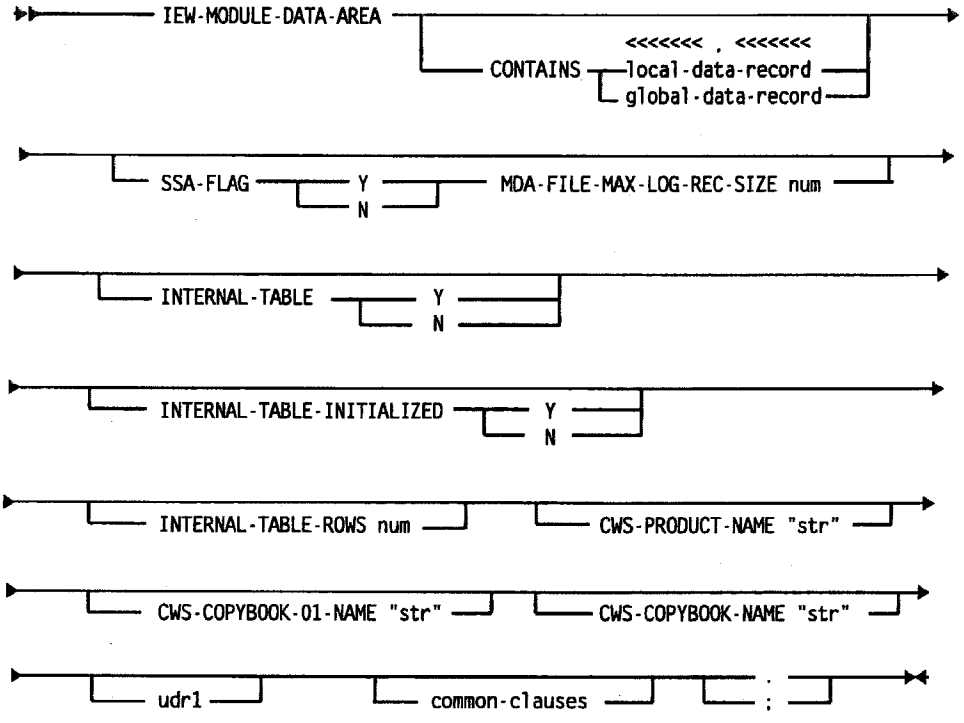
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-MODULE-DATA-AREA

ADW/IEW Design Workstation MODULE-DATA-AREA member type.

Syntax



where

local-data-record is an IEW-LOCAL-DATA-RECORD member name

global-data-record is an IEW-GLOBAL-DATA-RECORD member name

num is a positive integer

"str" is data to be imported from, or exported to, ADW/IEW

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-ORGANISATIONAL-UNIT

ADW/IEW Planning Workstation ORGANISATIONAL-UNIT member-type. An administrative sub-division of the enterprise.

CAUSES
Records the problems caused by this ORGANISATIONAL-UNIT

AFFECTS
Contains a list of the enterprise concerns - CRITICAL-SUCCESS-FACTORS, CRITICAL-ASSUMPTIONS, GOALS, PROBLEMS, and INFORMATION-NEEDS - cited by this ORGANISATIONAL-UNIT

CONTAINS
Must contain the name of every ORGANISATIONAL-UNIT which this ORGANISATION-UNIT co-ordinates.

LOCATION
States the location within the enterprise of this object by naming the appropriate LOCATION members.

INFLUENCES
Specifies the ORGANISATIONAL-UNITS managed by this ORGANISATIONAL-UNIT.

HAS
This clause contains the names of the SUBJECT-AREAS, DATA-COLLECTIONS, ENTITY-TYPES, FUNCTIONS, MECHANISMS, or PROCESSES that this ORGANISATIONAL-UNIT is responsible for.

SUPPORTS
Contains the names of one or more PROJECTS sponsored by this ORGANISATIONAL-UNIT

MANAGER
The manager identifies the name of the person that is responsible for ensuring that the role of that organisational unit is understood and fulfilled.

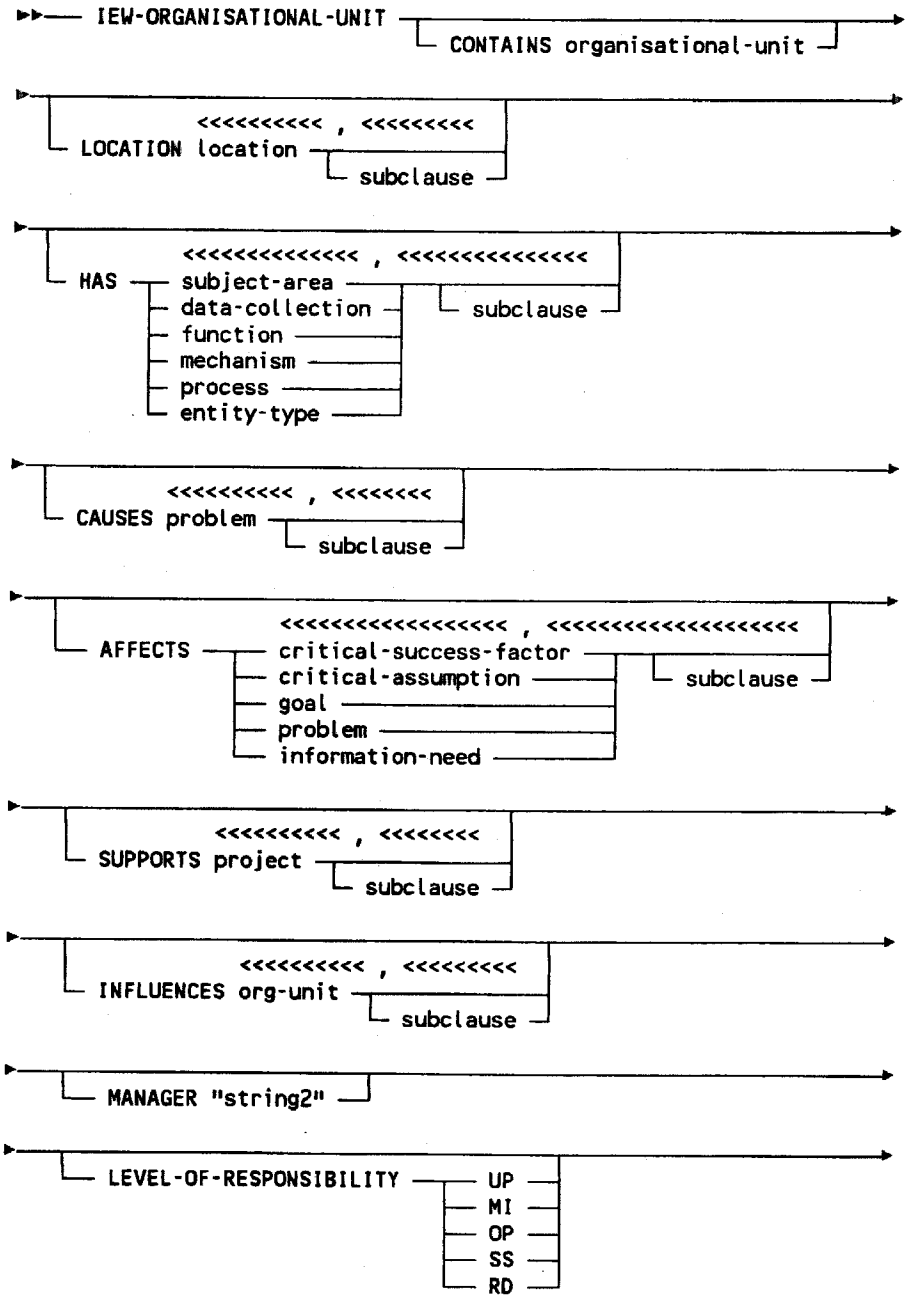
IEW-ORGANISATIONAL-UNIT

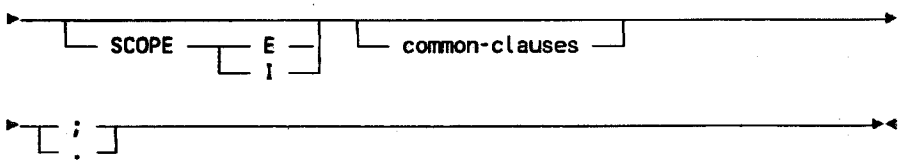
LEVEL-OF-RESPONSIBILITY
Indicates decision types and span of control
UP - UPPER MANAGEMENT
MI - MIDDLE MANAGEMENT
OP - OPERATIONAL MANAGEMNET
SS - SUPPORT STAFF
RD - RESEARCH DEVELOPMENT MANAGEMENT

SCOPE
E - External
I - Internal

IEW-ORGANISATIONAL-UNIT

Syntax



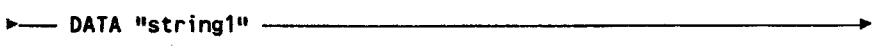


where

organisational-unit is an IEW-ORGANISATIONAL-UNIT member name

location is an IEW-LOCATION member name

subclause is:



where **"string1"** is the data to be imported from, or exported to, ADW/IEW

subject-area is an IEW-SUBJECT-AREA member name

data-collection is an IEW-DATA-COLLECTION member name

function is an IEW-FUNCTION member name

mechanism is an IEW-MECHANISM member name

process is an IEW-PROCESS member name

entity-type is an IEW-ENTITY-TYPE member name

problem is an IEW-PROBLEM member name

critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member name

critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

goal is an IEW-GOAL member name

IEW-ORGANISATIONAL-UNIT

information-need is an IEW-INFORMATION-NEED member name

"string2" is a character string, with a maximum length of 32 characters

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-PARAMETER

ADW/IEW Design Workstation PARAMETER member type.

CONTAINS
Must contain the name of every GLOBAL-DATA-STRUCTURE or LOCAL-DATA-STRUCTURE this PARAMETER is composed of.

SEE
Contain the name of any data-structures this PARAMETER references.

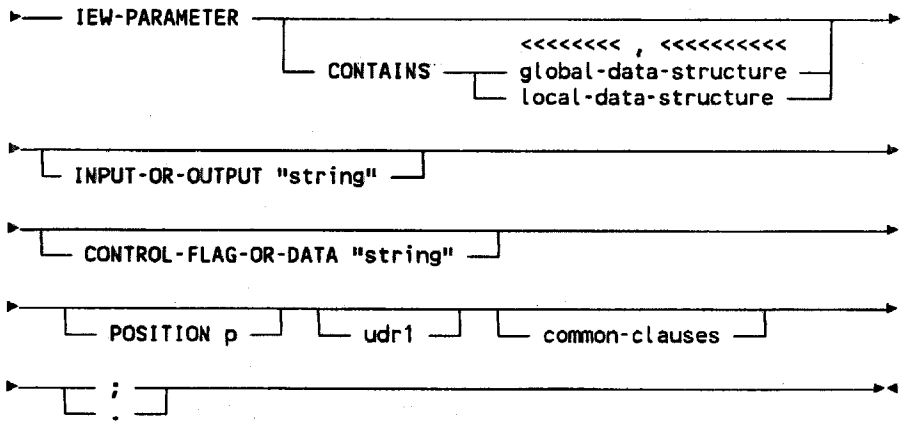
INPUT-OR-OUTPUT
Indicates what type of information the parameter will hold

CONTROL-FLAG-OR-DATA
Indicates whether this parameter is a control-flag or data.

POSITION
Position of the parameter as passed to the calling module.

IEW-PARAMETER

Syntax



where

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

"string" is the data to be imported from, or exported to, ADW/IEW

p is an integer

udr1 is as defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-PROBLEM

ADW/IEW Planning Workstation PROBLEM member type.
A situation or an issue that presents uncertainty, complication, unnecessary complexity or difficulty to the enterprise.

AFFECTS
Contains a list of the planning objects affected by this PROBLEM.

CAUSES
Records the problems caused by this PROBLEM.

HAS
This clause contains the names of the PROBLEMS that constitute this PROBLEM.

BEGIN-TIME
Date from which planning entity is effective
in the form MMM-YYYY.
The past, present or future date a critical assumption is initially effective.

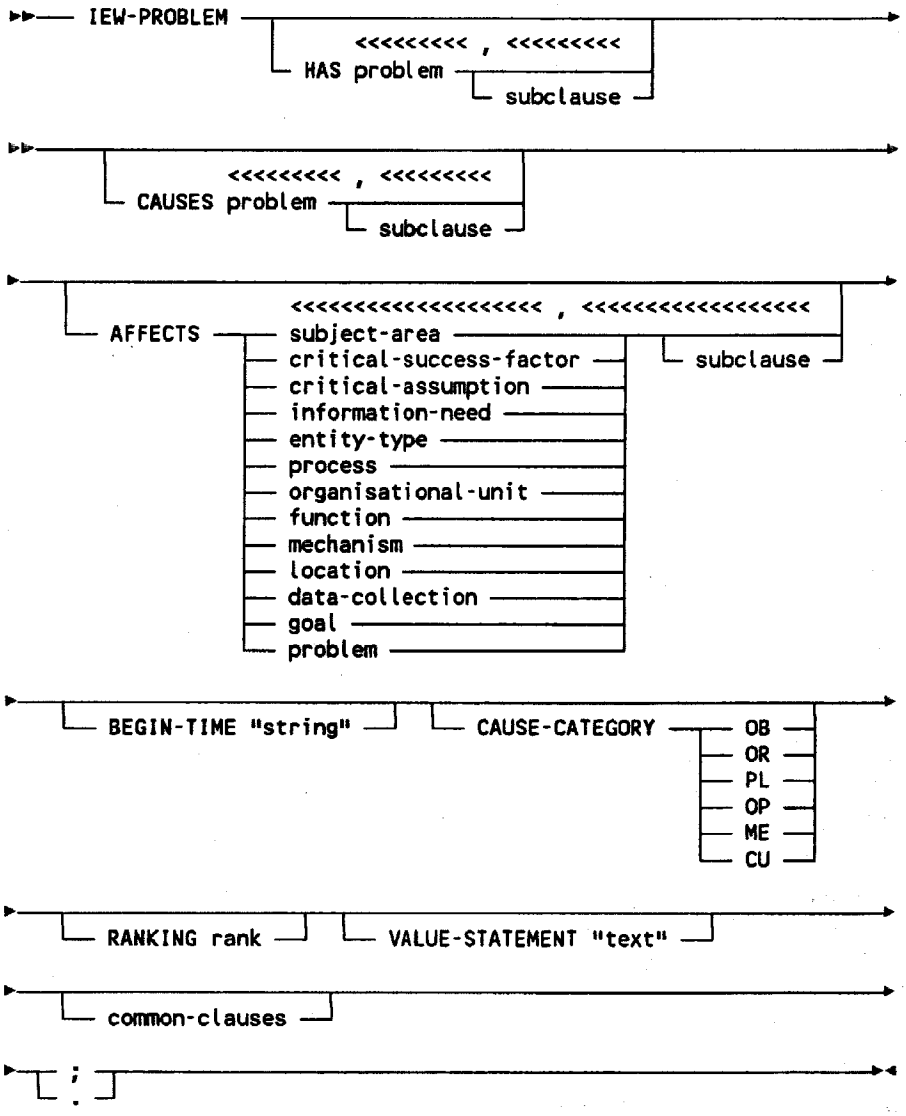
CAUSE-CATEGORY
Indicates classification of a problem's cause:
OB - Objectives
OR - Organisation
PL - Planning
OP - Operational Environment
ME - Measurement and control
CU - Current information system

RANKING
Relative importance of Planning Object
An integer from 1-9999
How important is this critical assumption relative to other critical assumptions of the enterprise?

VALUE-STATEMENT
Description of the perceived value of solving the problem, that is, a list of benefits realised when the problem is solved

IEW-PROBLEM

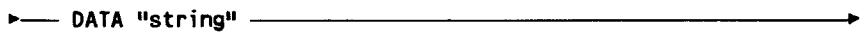
Syntax



where

problem is an IEW-PROBLEM member name

subclause is:



where **"string"** is the data to be imported from, or exported to, ADW/IEW

subject-area is an IEW-SUBJECT-AREA member name

critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member name

critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

information-need is an IEW-INFORMATION-NEED member name

entity-type is an IEW-ENTITY-TYPE member name

process is an IEW-PROCESS member name

organisational-unit is an IEW-ORGANISATIONAL-UNIT member name

function is an IEW-FUNCTION member name

mechanism is an IEW-MECHANISM member name

location is an IEW-LOCATION member name

data-collection is an IEW-DATA-COLLECTION member name

goal is an IEW-GOAL member name

"string" is as defined above

rank is an unsigned numeric value in the range 1 to 9999

"text " is up to 32767 delimited strings, each string having a maximum length of 60 characters

IEW-PROBLEM

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-PROCESS

ADW/IEW Analysis Workstation PROCESS member type.
An activity that is repeatedly executed within the enterprise, each execution of which produces a specific kind of effect on entities, or information about entities, of specific types.
A process may be described in an action diagram.

CAUSES
Records the problems caused by this PROCESS and the processes preceded by this PROCESS.

HAS
This clause contains the names of the PROCESSES or SEQUENTIAL-PROCESSES that constitute this PROCESS.

CONTAINS
Must contain the name of every JUNCTION which is a part of this PROCESS.

Must contain the name of every EXTERNAL-AGENT with which this PROCESS interacts.

SEE
List of ENTITY-TYPES, ATTRIBUTE-TYPES or RELATIONSHIP-TYPES that are involved in this PROCESS.

RELIES-ON
Lists the DATASTORES which form the scope of this PROCESS.

LOCATION
States the location within the enterprise of this object by naming the appropriate LOCATION members.

SUPPORTS
Must contain the name of every planning object which is a part of this PROCESS.

CENTRAL-TRANSFORM
Is this object transformed centrally, Y or N.

IEW-PROCESS

DECLARED-ROOT
Y - yes
N - no
Declaring an object a root states your intention of viewing the object as a separate decomposition diagram.
It does not automatically create a separate decomposition diagram.

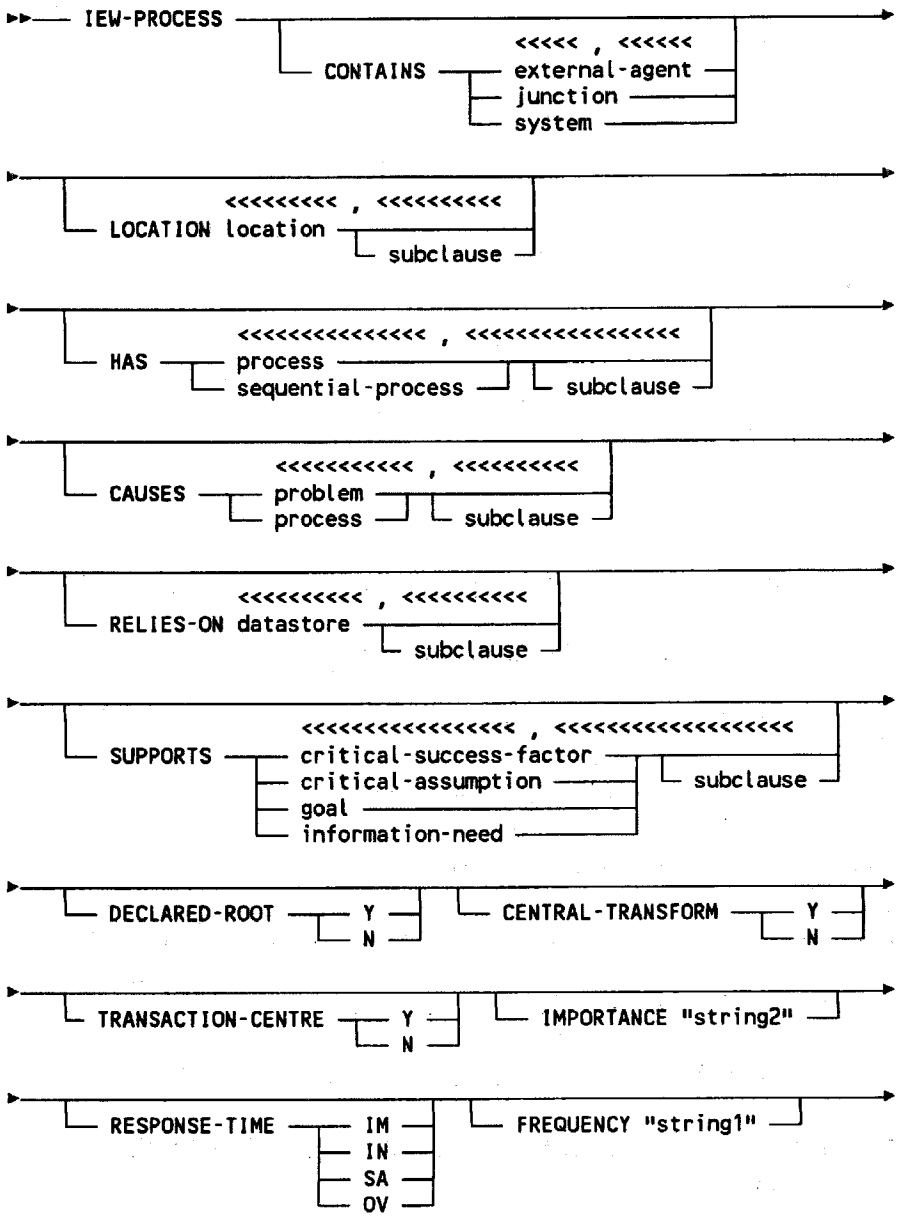
TRANSACTION-CENTRE
Is this object a transaction centre, Y or N.

IMPORTANCE
Relative importance of modelling or implementing the Process
Values H(igh) M(edium) L(ow) or integer from 1 - 999

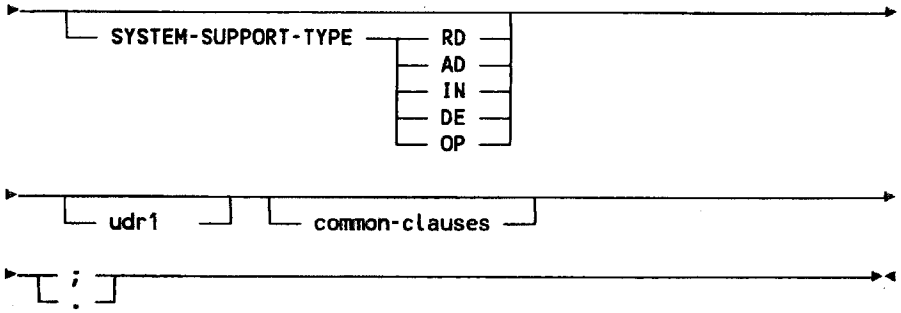
RESPONSE-TIME
IM - Immediate (3 secs or less)
IN - Interactive (1 minute or less)
SA - Same day
OV - Overnight

SYSTEM-SUPPORT-TYPE
Type of support

Syntax


IEW-PROCESS
CONTAINS
<<<<< , <<<<<<
external-agent
junction
system
LOCATION location
<<<<<<<<< , <<<<<<<<<<
subclause
HAS
<<<<<<<<<<<<<< , <<<<<<<<<<<<<<<<<<
process
sequential-process
subclause
CAUSES
<<<<<<<<<< , <<<<<<<<<<
problem
process
subclause
RELIES-ON datastore
<<<<<<<<<< , <<<<<<<<<<
subclause
SUPPORTS
<<<<<<<<<<<<<<<< , <<<<<<<<<<<<<<<<<<<<
critical-success-factor
critical-assumption
goal
information-need
subclause
DECLARED-ROOT
Y
N
CENTRAL-TRANSFORM
Y
N
TRANSACTION-CENTRE
Y
N
IMPORTANCE "string2"
RESPONSE-TIME
IM
IN
SA
OV
FREQUENCY "string1"

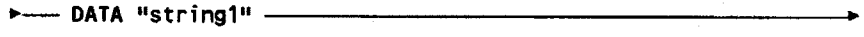
IEW-PROCESS



where

external-agent is an IEW-EXTERNAL-AGENT member name

subclause is



where **"string1"** is the data for import from, or export to, ADW/IEW

junction is an IEW-JUNCTION member name

system is a SYSTEM member name

location is an IEW-LOCATION member name

process is an IEW-PROCESS member name

sequential-process is an IEW-SEQUENTIAL-PROCESS member name

problem is an IEW-PROBLEM member name

datastore is an IEW-DATASTORE member name

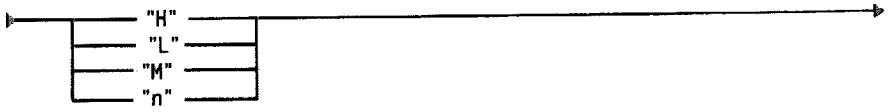
critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member name

critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

goal is an IEW-GOAL member name

information-need is an IEW-INFORMATION-NEED member name

"string2" is



"n" is an integer in the range "1" to "999"

udr1 is as defined in Appendix 2

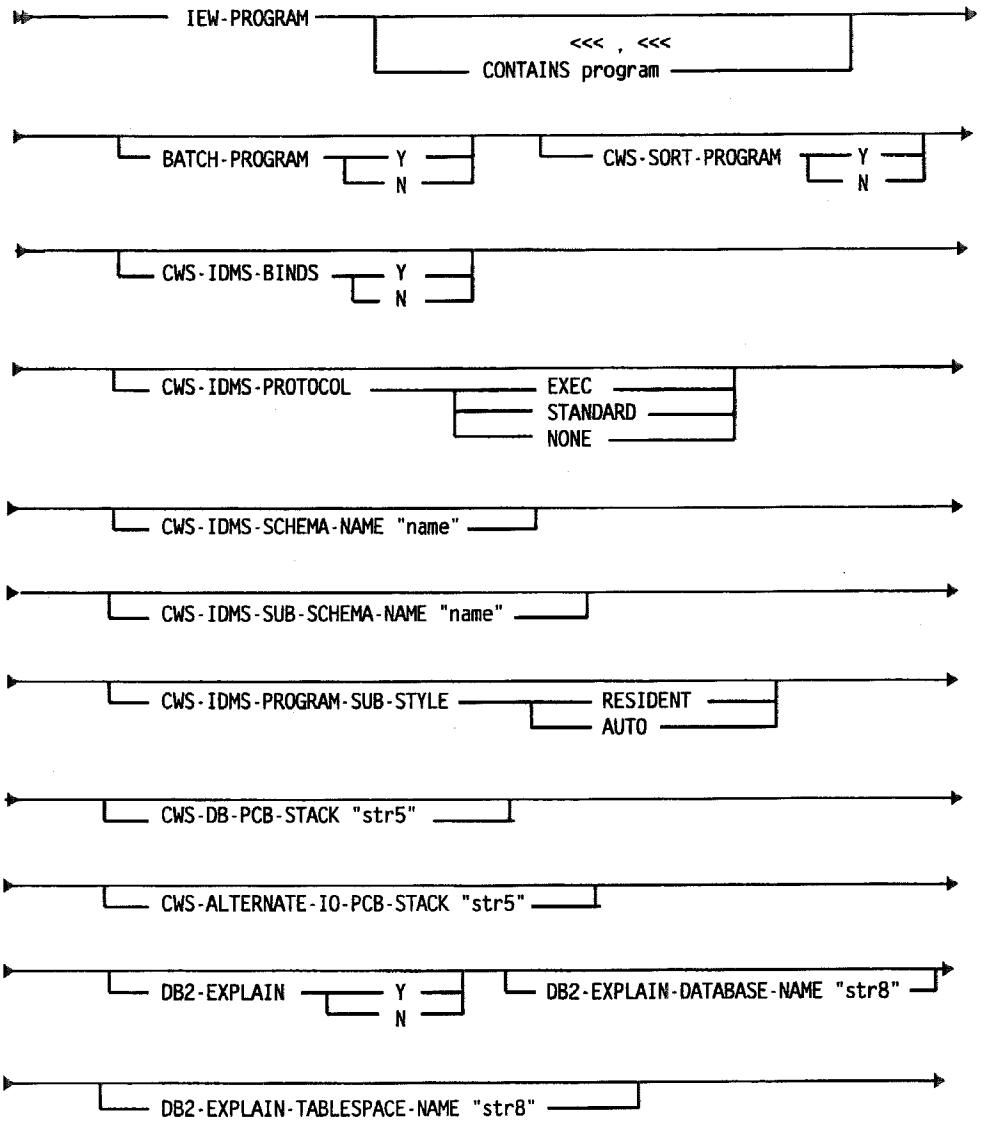
common-clauses are any of the clauses available to all member types.

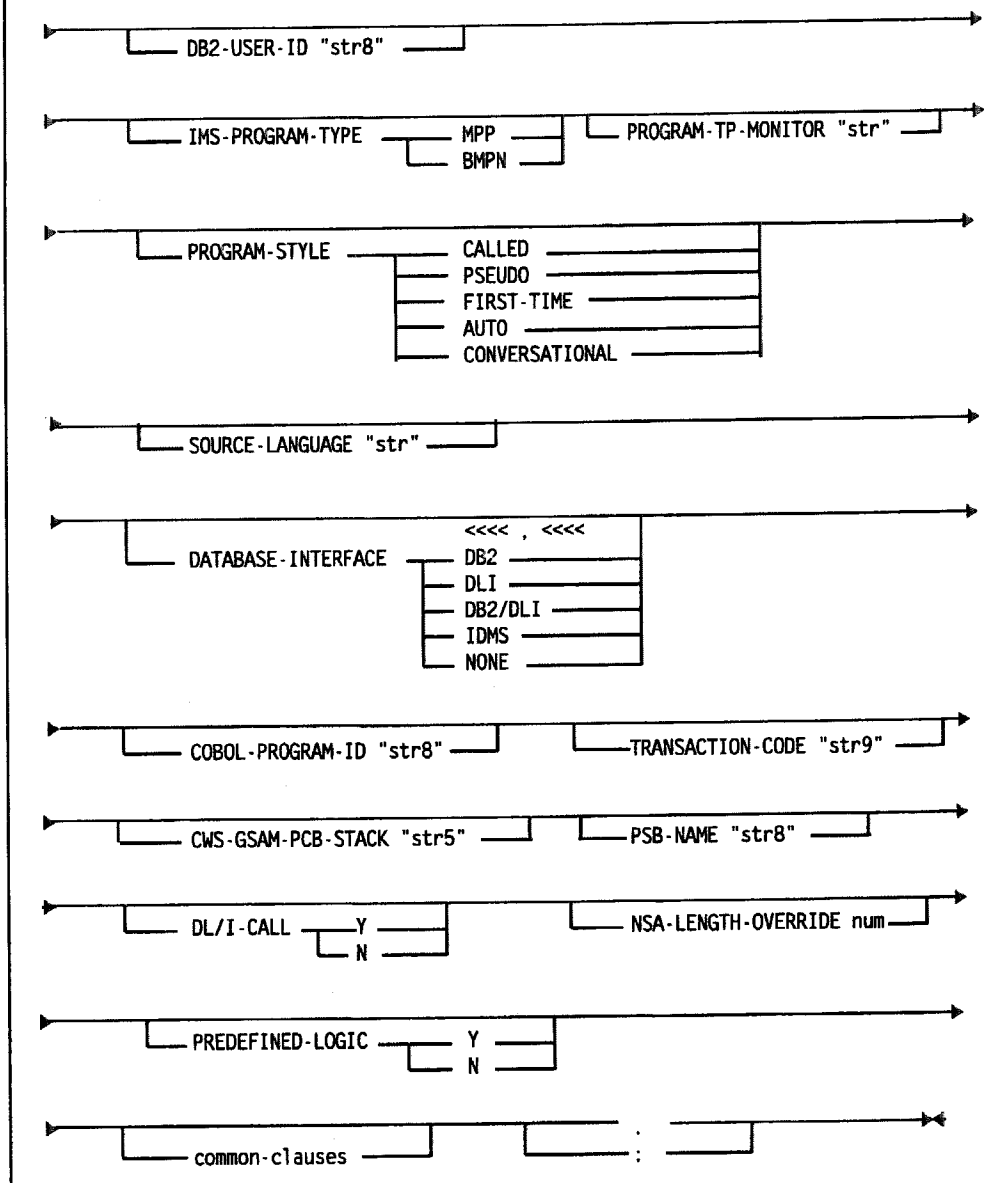
Refer to Appendix 2 for details of the common clauses.

IEW-PROGRAM

ADW/IEW Design Workstation PROGRAM member type.

Syntax




DB2-USER-ID "str8"
IMS-PROGRAM-TYPE
MPP
BMPN
PROGRAM-TP-MONITOR "str"
PROGRAM-STYLE
CALLED
PSEUDO
FIRST-TIME
AUTO
CONVERSATIONAL
SOURCE-LANGUAGE "str"
DATABASE-INTERFACE
<<<< , <<<<
DB2
DLI
DB2/DLI
IDMS
NONE
COBOL-PROGRAM-ID "str8"
TRANSACTION-CODE "str9"
CWS-GSAM-PCB-STACK "str5"
PSB-NAME "str8"
DL/I-CALL
Y
N
NSA-LENGTH-OVERRIDE num
PREDEFINED-LOGIC
Y
N
common-clauses
:
:

where

program is an IEW-PROGRAM member name

num is a positive integer

"**name**" is data to be imported from, or exported to, ADW/IEW

"**str**" is data to be imported from, or exported to, ADW/IEW

"**str5**" is a 5 character string

"**str8**" is an 8 character string

"**str9**" is a 9 character string

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

IEW-PROJECT

ADW/IEW Planning Workstation PROJECT member type. A systematic plan of action, requiring concerted effort, to model or implement a part of the enterprises information system.

ASSIGNED-TO
Contains the name of the GOAL, CA, CSF, INFORMATION-NEED or PROBLEM that this PROJECT addresses.

CONTAINS
This clause contains the names of the PROJECTS that constitute this PROJECT.

HAS
This clause contains the names of the SUBJECT-AREAS, DATA-COLLECTIONS, ENTITY-TYPES, FUNCTIONS, MECHANISMS, PROCESSES or INFORMATION-NEEDS that this PROJECT includes.

LOCATION
States the location within the enterprise of this object by naming the appropriate LOCATION members.

RELIES-ON
Lists the PROJECTS that this PROJECT itself requires.

ACTUAL-END-DATE
Date when actually completed
IEW format MM/DD/YYYY

ACTUAL-START-DATE
Date when actually started
IEW format MM/DD/YYYY

ESTIMATED-MAN-MONTHS
Real number greater than zero
A subjective estimate of how long the project will run in man-months.

ESTIMATED-COST
An integer signifying the estimated resource cost.

PRIORITY
Priority assigned to project
Values H(igh) M(edium) L(ow) or integer from 1 - 999

RANKING
Relative importance of Planning Object
An integer from 1-9999
How important is this critical assumption relative to other critical assumptions of the enterprise?

RETURN-ON-INVESTMENT
An indication of the Return on Investment.

RISK
Degreee of risk assigned to project
Values H(igh) M(edium) L(ow) or integer from 1 - 999

SCHEDULED-START-DATE
Date when supposed to be started
IEW format MM/DD/YYYY

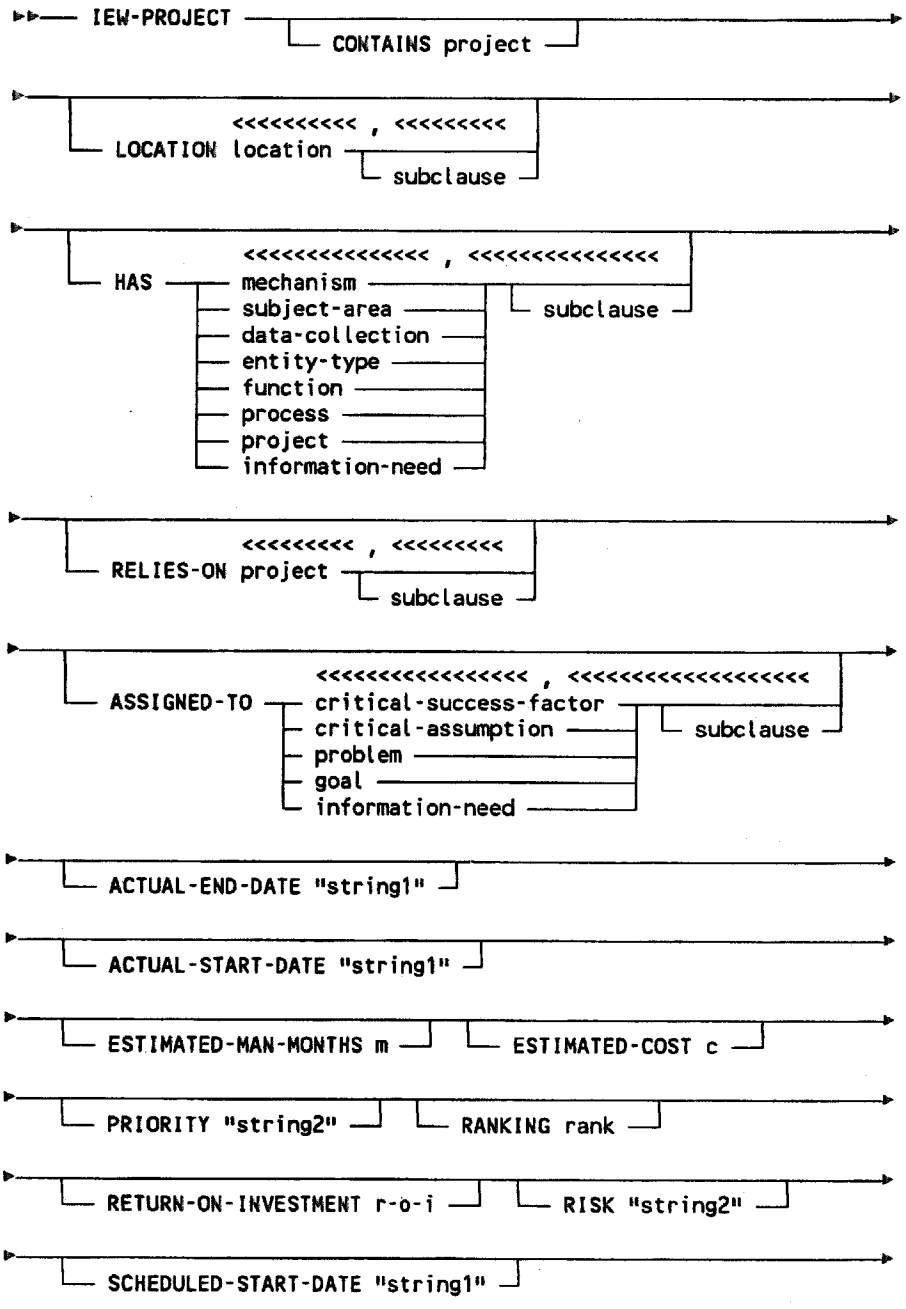
SCHEDULED-END-DATE
Date when supposed to be finished
IEW format MM/DD/YYYY

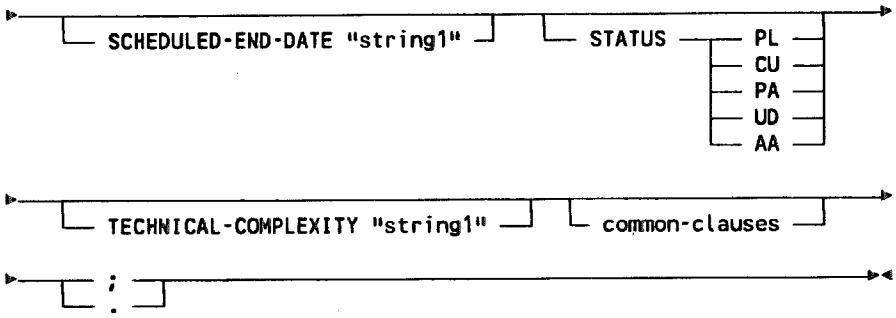
STATUS
Code used to describe the state of the members types.
The codes have the following meanings:
PL - PLANNED
CU - CURRENT
PA - PAST
UD - UNDER DEVELOPMENT
AA - CREATED BY AFFINITY ANALYSIS

TECHNICAL-COMPLEXITY
Technical complexity of project
Values H(igh) M(edium) L(ow) or integer from 1 - 999

IEW-PROJECT

Syntax





where

project is an IEW-PROJECT member name

location is an IEW-LOCATION member name

subclause is:

where **"string1"** is the data for import from, or export to, ADW/IEW

mechanism is an IEW-MECHANISM member name

subject-area is an IEW-SUBJECT-AREA member name

data-collection is an IEW-DATA-COLLECTION member name

entity-type is an IEW-ENTITY-TYPE member name

function is an IEW-FUNCTION member name

process is an IEW-PROCESS member name

project is an IEW-PROJECT member name

problem is an IEW-PROBLEM member name

information-need is an IEW-INFORMATION-NEED member name

IEW-PROJECT

critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member name

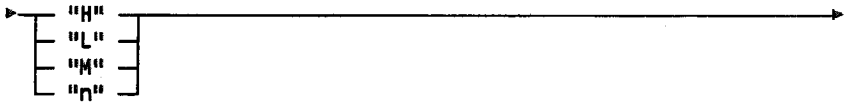
critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

goal is an IEW-GOAL member name

"string1" is as defined above

m is an unsigned integer

"string2" is



"n" is an integer in the range "1" to "999"

rank is an unsigned integer in the range 1 to 9999

r-o-i and **c** are integers in the range 1 to 99,999,999

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-PSB

ADW/IEW Design Workstation PSB member type.

CMPAT
Is an I/O PCB always required for compatibility between batch DL/1 and BMP/MSG processing, YES or NO.

IOASIZE
An integer (0-256000) to indicate the maximum I/O area size, if unspecified the ACBGEN utility will calculate this value.

MAXQ
An integer signifying the maximum number of Qx command code database calls which may be issued betweed synchronisation points, if exceeded an abend occurs.

SSA-SIZE
An integer specifying the maximum total length of all SSAs to be used by the application program

OLIC
Specify YES if it is required to authorise users of this PSB for OnLine Image Copy or Surveyor utilities (Invalid for CICS, GSAM, or HSAM database)

IOEROPN
An integer (0-4095) signifying condition code to be returned to operating system when program terminates normally after encountering database I/O errors in a batch region.

IOEROPM
Wait for operator response flag (character string)
Only valid if IOEROPN has been specified

LANGUAGE
The name of the computer language in which the member is programmed.

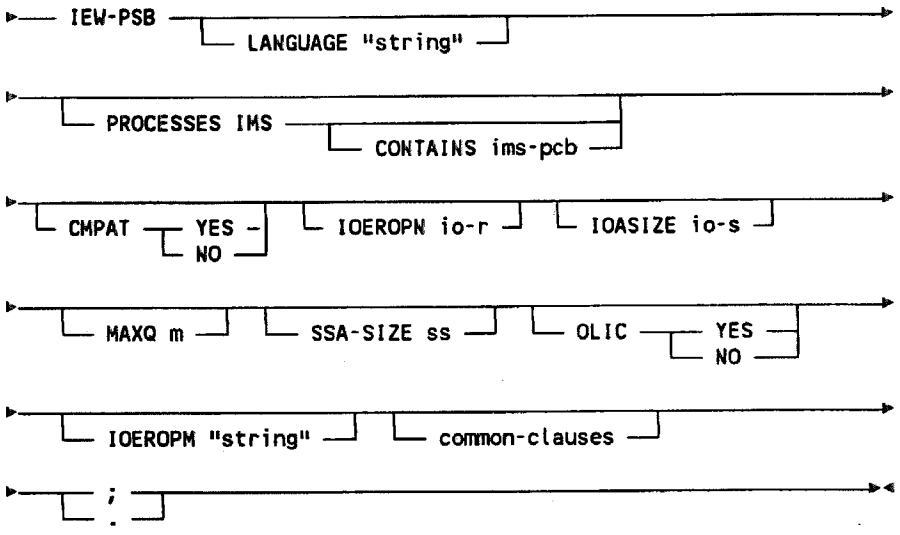
PROCESSES
Contains the name of each database processed through this process-view. Each database whose processing is described in the DESCRIPTION clause of this member must be named in this clause.
A separate PROCESSES attribute may specify each database processing requirement in free format.
NB; For IMS processing the PROCESSES IMS CONTAINS attribute should be used to specify the constituent database PCB member names.

PROCESSES IMS CONTAINS
Contains the name of each database PCB referenced through this process-view.

SEGMENT-SEARCH-ARGUMENTS
May contain a list of Segment Search Arguments (SSAa) for sensitive segments contained by the Program Communication Blocks (PCBs) which are referenced by this program.
Multiple USED-IN clauses may be specified where there are many SSAs for a segment.

SSAS
Will only appear for existing PROGRAM members if the keyword SSAS has been specified as an alternative to SEGMENT-SEARCH-ARGUMENTS.

Syntax



where

"string" is data for import from, or export to, ADW/IEW

ims-pcb is a PCB member name

io-r, io-s, m, and ss are unsigned integers

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-RELATION

ADW/IEW Design Workstation RELATION member type.
A record type in a relational database.

SEE
List of LOCAL and GLOBAL-DATA-STRUCTURES describing this data-area. May also include other RELATIONS.

AS
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

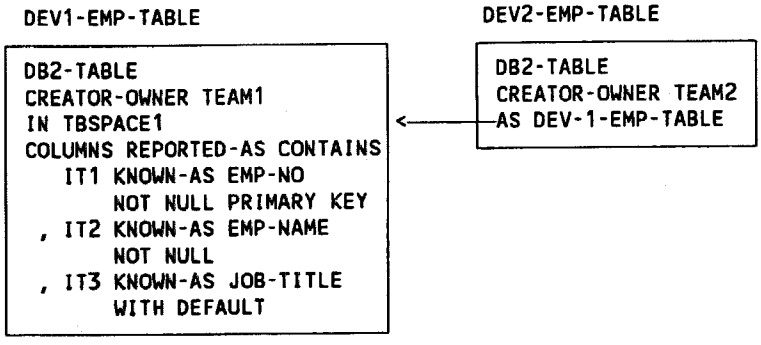
- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

For example, if different development teams require versions of the same table, you can fully define one member, DEV1-EMP-TABLE, then use the AS clause in the second member, DEV2-EMP-TABLE.



The generated tables both belong to TBSPACE1 and both have the same columns.

When you generate an SQL CREATE statement for DEV2-EMP-TABLE, the CREATOR-OWNER clause in DEV1-EMP-TABLE is not extracted via the AS clause because DEV2-EMP-TABLE has its own CREATOR-OWNER clause defined.

CREATOR-OWNER
To define the owner of an object, enter:

```
CREATOR-OWNER user
```

user is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the object.
On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member.
The CREATOR-OWNER clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in an export to DB2 panel.

Note: the DB2-USER or SQL-USER member named in the CREATOR-OWNER clause is used to generate user-qualified names for tables, views and aliases. If the DB2-USER has a LOCATION clause defined, and the DB2 profile is set to three-part name generation, location and user-qualified names are generated for tables and views. The location-qualifier can be overridden by a LOCATION clause defined in an export to DB2 panel.

IN
To define the table space in which the table is created, enter:

```
IN tbspace
```

tbspace is the name of a DB2-TBSPACE member.

On encoding the member specified is checked to ensure it is a DB2-TBSPACE member.

For the successful generation of SQL statements the DB2-TABLE definition must include an IN clause, naming a DB2-TBSPACE member. The DB2-TBSPACE must include an IN clause, naming a DB2-DATABASE. This chain of relationships defines the database to which the table space, and table belong.

COLUMNS
To specify which form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns in the table, enter:

```
COLUMNS form-keyword
```

form-keyword is one of the following:

ENTERED-AS
HELD-AS
REPORTED-AS
DEFAULTED-AS

The form keyword that you define in the COLUMNS clause applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.

For example, a DB2-TABLE member containing the following lines:

```
COLUMNS ENTERED-AS
CONTAINS ITEM1, ITEM2
```

refers to the two ITEM members:

ITEM1

```
ITEM
HELD-AS BINARY 10
ENTERED-AS CHAR 5
REPORTED-AS FLOAT 9
```

ITEM2

```
ITEM
ENTERED-AS DECIMAL 4.2
DEFAULTED-AS FLOAT 7
```

The ENTERED-AS form keyword in the DB2-TABLE definition specifies that the ENTERED-AS form description from both ITEMs is used to define the data type of columns. Therefore the column generated from ITEM1 has a data type of CHAR, and the column generated from ITEM2 has a data type of DECIMAL.
If you do not specify a form keyword then the DEFAULTED-AS form description is used. Where the ITEM or GROUP has no DEFAULTED-AS form description defined, then the MANAGER Products defaults apply. For further information refer to the DATAMANAGER Source Language Generation manual, section 2.2.3, remarks 39 to 41.

To specify the ITEM or GROUP members that define columns, enter:

```
CONTAINS member-list
```

member-list is the name of one or more ITEM or GROUP members, separated by commas, each representing a single column.
On encoding, the members specified in the CONTAINS clause are checked to ensure that they are either ITEMs or GROUPs.
Duplicate column names are not permitted by DB2, therefore column names are checked on generation to ensure that no duplicates are present.

Each ITEM can define up to 15 form descriptions. To define which of the form descriptions you want to use, enter:

```
CONTAINS item version
```

item is the name of an ITEM member.
version is an integer in the range 1 to 15, and defines the form description version that you want to use.

For example:

```
HELD-AS CONTAINS STOCK-LIST 3
```

defines that the third HELD-AS form description in the ITEM member STOCK-LIST is used as the column data type.

To define a set of columns with identical attributes, using the same ITEM or GROUP member, enter:

```
CONTAINS (integer) member
```

integer is the number of columns to be derived from the member, within brackets.

member is the name of an ITEM or GROUP member.

To resolve duplicated names on generation of an SQL statement, column names are automatically suffixed by an underscore and a number, the first by_1, the second by_2 and so on.
For example:

```
CONTAINS (4) STOCK-LIST
```

generates the four columns STOCK_LIST_1, STOCK_LIST_2, STOCK_LIST_3 and STOCK_LIST_4. The generated attributes, such as data type, are the same for each of the four columns.

When a column is derived from a GROUP containing several ITEMs, the data type of the column is taken as CHAR. The maximum number of characters allowed in the column is calculated from the combined field lengths of the form descriptions defined in each ITEM. However, where a DB2 command applied to the DB2-TABLE specifies the EXPAND keyword, then each ITEM within a GROUP generates a separate column.

You can define additional attributes for a table's column(s) using sub-clauses within the CONTAINS clause. If the CONTAINS clause defines a column set or an expanded GROUP, the generated column attributes apply to all the columns in the set.
To specify the contents of the column as bit (binary) data, enter:

```
FOR-BIT-DATA
```

Note: the FOR-BIT-DATA keyword is only valid where columns have the data-type of CHAR, VARCHAR or LONG VARCHAR specified in the form description.

To define that columns cannot contain a null value, and are set to a default value by DB2 , enter:

```
WITH-DEFAULT
```

The WITH-DEFAULT keyword generates NOT NULL WITH DEFAULT in SQL statements.

To specify that a column cannot contain a null value, enter:

```
NOT-NULL
```

NOT-NULL and WITH-DEFAULT are mutually exclusive. If you specify both the member will not encode.

Columns with a data type of CHAR can also have an associated FIELDPROC procedure, provided WITH-DEFAULT is not defined. To define a FIELDPROC, enter:

```
FIELDPROC process
```

process is the name of a SYSTEM, MMR-SYSTEM, PROGRAM or MODULE member, and represents a field procedure that exists in DB2.

To define the parameters passed to the field procedure, enter:

```
CONSTANT 'list'
```

list is one or more parameters separated by commas. The list must be no more than 254 characters, within delimiters. For example:

```
CONTAINS IT-INCOMING NOT-NULL FIELDPROC MOD-XT3
                                        CONSTANT
"CONST4-3, CONST4-4"
```

defines that the ITEM member IT-INCOMING generates a column that:

- cannot contain a null value, and
- uses the process member MOD-XT3, passing the parameters CONST4-3 and CONST 4-4.

To define that a column forms the primary key of a table, enter:

```
PRIMARY-KEY
```

A primary key can comprise a maximum of 16 columns. For the successful generation of an SQL CREATE TABLE statement, columns defined as PRIMARY-KEY must also be defined as either NOT-NULL or WITH-DEFAULT.

Where your primary key consists of several columns, the sequence in which columns constitute the key is assumed to be the sequence in which they are defined, unless you specify the key position.

To define the position of the column in a multi-column primary-key, enter:

```
PRIMARY-KEY kpos
```

kpos is an integer in the range 1 to 16.
For example:

```
CONTAINS
  DEPT-NO
, EMP-NO NOT-NULL PRIMARY-KEY 2
, JOB-TITLE
, EMP-NAME NOT-NULL PRIMARY-KEY 1
```

defines that the primary key of this table is EMP-NAME followed by EMP-NO, rather than EMP-NO followed by EMP-NAME.
To define that the entries in a primary key column are sorted in ascending key order in the index, enter:

```
PRIMARY-KEY ASC
```

To define that the entries in a primary key column are sorted in descending key order in the index, enter:

```
PRIMARY-KEY DESC
```

To define that a column is to contain a comment enter:

```
DB2-COMMENT 'comment'
```

To define that a column is to contain a label enter:

```
DB2-LABEL 'label'
```

To define a referential constraint for a table, enter:

```
CONSTRAINT constraint
```

constraint is one, several or all of the following:

- the NAMED clause defines a constraint name
- the FOREIGN-KEY clause specifies one or more columns to form the foreign key
- the REFERENCES clause specifies the table being referenced
- the DELETE clause defines the associated DELETE rule.

You may define any number of referential constraints for a table. Each referential constraint requires its own CONSTRAINT clause. Each CONSTRAINT clause must include the FOREIGN-KEY and REFERENCES clauses for the successful generation of SQL CREATE statements.

You can name a referential constraint without specifying the column(s) that form the foreign key. Thus you can set up DB2-TABLE definitions with named referential constraints between them before deciding on the contents of the tables. This feature is useful in a top-down approach to database design.

To define a constraint name, enter:

```
NAMED constraint-name
```

constraint-name is the name, of no more than 8 characters, by which the referential constraint is known to DB2.
Each constraint name must be unique within a table. If you do not specify a constraint name, a default name is generated by DB2.

To define the foreign key for a constraint, enter:

```
FOREIGN-KEY
```

followed by clauses that define the columns that comprise the foreign key, that is:

- specifying that one or more ITEMs and/or GROUPs each define a single column, see item 7
- specifying that each ITEM contained in a GROUP defines one column, see item 10.

The columns comprising the foreign key must already be defined in the CONTAINS clause, as the foreign key must exist as a column of the table.

You can name the foreign key column using the KNOWN-AS clause. When you generate an SQL statement the name defined in the KNOWN-AS clause is checked against the columns generated. The column name and foreign key name must be the same.
For example, the columns of a table are generated from an expanded GROUP member. However the foreign key comprises only one of the table's columns.

The DB2-TABLE definition includes the clauses:

```
CONTAINS MANY-ITEMS EXPAND
CONSTRAINT FOREIGN-KEY ITEM4 KNOWN-AS COL-4
```

Although the CONTAINS clause does not name the member ITEM4, the GROUP, MANY-ITEMS, includes the clauses:

```
CONTAINS
   ITEM1 KNOWN-AS COL-1
 , ITEM2 KNOWN-AS COL-2
 , ITEM3 KNOWN-AS COL-3
 , ITEM4 KNOWN-AS COL-4
```

The generated table has four columns, COL_1, COL_2, COL_3 and COL_4. The foreign key column is COL_4.

You can clarify correspondence between members in the repository where:

- foreign key column names differ from corresponding primary key column names
- different ITEM and/or GROUP members represent the foreign key columns in one table and the corresponding primary key columns in another table

except where the foreign key is defined using an expanded GROUP.

To specify the column(s) in another table to which the foreign key refers, enter:

```
MEMBER member
```

member is the name of an ITEM or GROUP member defining the column(s).

Where the name of a foreign key column(s) is different from its name in another table, you can optionally specify the local name of the column using the KNOWN-AS clause (see item 9).

To define the table to which a foreign key refers, enter:

```
REFERENCES table
```

table is the name of a DB2-TABLE or SQL-TABLE member.

To successfully generate SQL statements:

- one REFERENCES clause must be defined for each foreign key specified
- the referenced DB2-TABLE must exist and include a valid CREATOR-OWNER clause.

To define the delete rule for the constraint, enter:

```
DELETE option
```

option is RESTRICT, CASCADE or SET-NULL.

The keywords are similar to those used by DB2 and they have the same meanings.

If you do not define a DELETE rule the DB2 default applies.

LIKE
In DB2, you may wish to create a table with the same column structure as an existing table, so that you can use it in production and development. To copy the columns of an existing table in DB2, enter:

```
LIKE member
```

member is the name of a DB2-TABLE, SQL-TABLE, DB2-VIEW or SQL-VIEW member.

When you generate SQL CREATE TABLE statements from a DB2-TABLE that includes a LIKE clause, the statement also contains a LIKE clause. When the statement is submitted to DB2 the table or view named in the LIKE clause must exist, or the column structure cannot be copied.

If you want to generate SQL DECLARE statements or host language data structures from a DB2-TABLE defined using a LIKE clause, you must also define an AS clause referring to the same member. For example:

```
LIKE TA-TOTAL-STOCK
AS TA-TOTAL-STOCK
```

defines that the table copies the generated columns of the DB2-TABLE TA-TOTAL-STOCK.

EDITPROC
To define an edit routine for the table, enter:

```
EDITPROC process
```

process is the name of a SYSTEM, MMR-SYSTEM, PROGRAM or MODULE member.

The member represents an edit routine that must exist in DB2. In DB2, the edit routine is invoked whenever a row in the table is is retrieved, updated or inserted.

VALIDPROC
To define a validation routine for the table, enter:

```
VALIDPROC process
```

process is the name of a SYSTEM, MMRSYSTEM, PROGRAM or MODULE member.

IEW-RELATION

The member represents a validation routine that must exist in DB2. In DB2, the validation routine receives an entire table row as input and may be used to control a subsequent INSERT, UPDATE or DELETE statement.

AUDIT
The auditing options for the table. Valid values are NONE, CHANGES and ALL.

CARDINALITY
Specifies the approximate number of rows which will be contained in the table.

DB2-COMMENT
To define a comment for an alias, table, view, or column, enter a string of no more than 254 characters.
When generated, lines of comment are concatenated to form a single string. To preserve spaces between words, insert a space after the last character of each continuing line.

For example, the DB2-TABLE named MANAGER-NUMBER has an owner of PERSONNEL and the following comment defined:

> This table contains the Manager number of every manager in each department.

The following SQL statement can be generated:

```
COMMENT ON TABLE PERSONNEL.MANAGER_NUMBER IS 'This table cont
ains the Manager number of every manager in each department'
```

In this example the word "contains" has been split due to the margins set in the DB2 profile.

This clause must be present for the successful generation of SQL COMMENT ON statements.

Note:

- for columns, the comment definition must follow the CONTAINS clause defining that column or group of columns

- for tables and views, the comment definition should precede the COLUMNS clause that defines the columns of the table.

DB2-LABEL
To define a label for an alias, table, view or column, enter a string of no more than 30 characters.

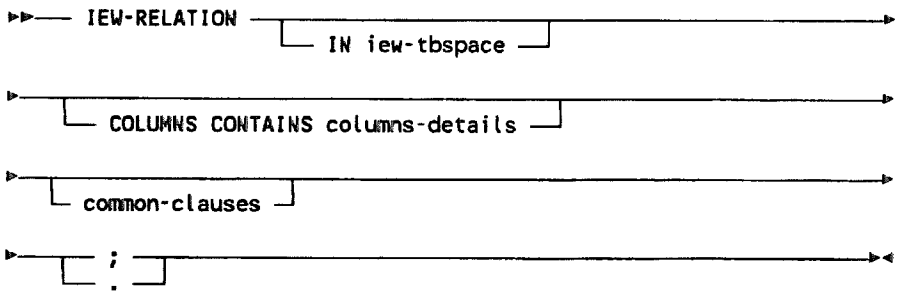
This clause must be present for the successful generation of SQL LABEL ON statements.

Note:

- for columns, the label definition must follow the CONTAINS clause defining that column or group of columns
- for tables and views, the label definition should precede the COLUMNS clause that defines the columns of the table.

IEW-RELATION

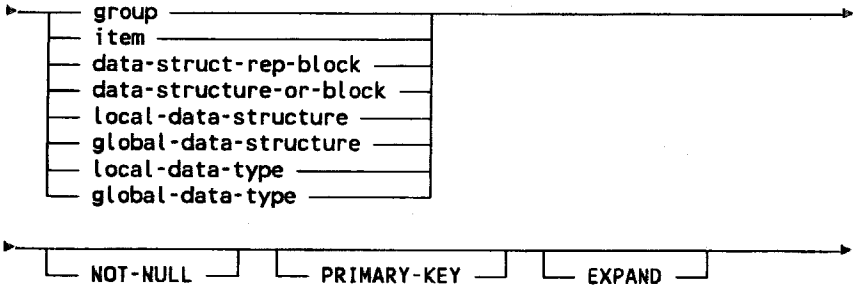
Syntax



where

iew-tbspace is an IEW-TBSPACE member name

columns-details are



where

group is a GROUP member name

item is an ITEM member name

data-struct-rep-block is an IEW-DATA-STRUCT-REP-BLOCK member name

data-structure-or-block is an IEW-STRUCTURE-OR-BLOCK member name

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

local-data-type is an IEW-LOCAL-DATA-TYPE member name

global-data-type is an IEW-GLOBAL-DATA-TYPE member name

common-clauses are any of the clauses available to all member types.

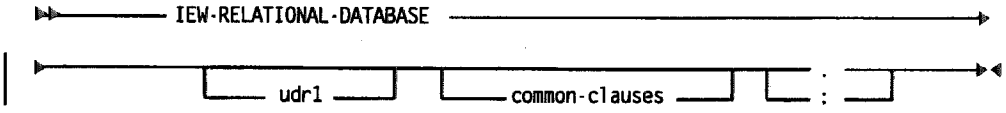
Refer to Appendix 2 for details of the common clauses.

Note: the EXPAND keyword is present by default but you can exclude it by tailoring Corporate Executive Routine MPDYIITAB0.

IEW-RELATIONAL-DATABASE

ADW/IEW Design Workstation RELATIONAL-DATABASE member type.

Syntax



where

udr1 is defined in Appendix 2

common-clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

This page is intentionally blank.

This page is intentionally blank.

IEW-RELATIONSHIP-TYPE

ADW/IEW Analysis Workstation RELATIONSHIP-TYPE member type. A kind of connection or interaction that can exist between two entities of a given type.

CONTAINS
Must contain the name of every ENTITY to which this relationship refers.

LR-MAX
An integer from 1-9999 or M (for "many")
The maximum number of instances of the right entity type that can have this type of relationship with each instance of the left entity type.

This cardinality constraint answers the question, "At most, how many right entities (instances of the entity type that is the right entity type in this relationship type) can be related to each left entity (an instance of the entity type that is the left entity type in this relationship type)?"

LR-MIN
An integer from 0-9999
The minimum number of instances of the right entity type that must have this type of relationship with each instance of the
left entity type.

This constraint answers the question, "At least, how many right entities (instances of the entity type that is the right entity type in this relationship type) must be related to each left entity (an instance of the entity type that is the left entity type in this relationship type)?"

RL-MAX
An integer from 1-9999
M (for "many")
The maximum number of instances of the left entity type that can have this type of relationship with each instance of the right entity type.

This cardinality constraint answers the question, "At most, how many left entities (instances of the entity type that is the left entity type in this relationship type) can be related to each right entity (an instance of the entity type that is the right entity type in this relationship type)?"

RL-MIN
an integer from 0-9999
The minimum number of instances of the left entity type that must have this type of relationship with each instance of the right entity type.

This cardinality constraint answers the question, "At least, how many left entities (instances of the entity type that is the left entity type in this relationship type) can be related to each right entity (an instance of the entity type that is the right entity type in this relationship type)?"

TO-FROM-NAME
1-32 characters
A character string that enables people to understand the kind of information that instances of this relationship type provide when instances of the right entity type are viewed as being described by instances of the left entity type.
Information planners should use a verb to name a relationship type.

For example, suppose a relationship type has been defined with "Student" as the left entity type and "Course" as the right entity type. A right to left name "enrolls" could be chosen, so that the relationship type can be read as "Course enrolls Student".

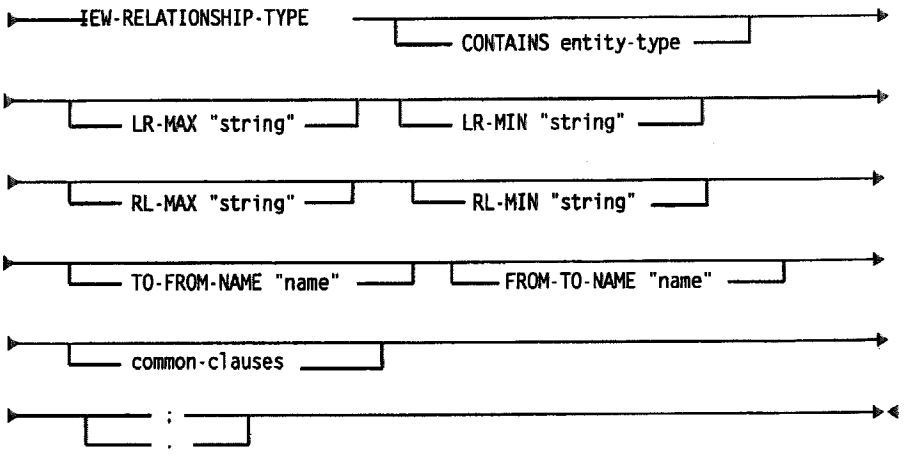
FROM-TO-NAME
1-32 characters
A character string that enables people to understand the kind of information that instances of this relationship type provide when instances of the left entity type are viewed as being described by instances of the right entity.

IEW-RELATIONSHIP-TYPE

Information planners should use a verb to name a relationship type.

For example, suppose that a relationship type has been defined with "Student" as the left entity type and "Course" as the right entity type. A left to right name of "is enrolled in" could be chosen, so that the relationship type can be read as "Student is enrolled in Course".

Syntax



where

entity-type is an IEW-ENTITY-TYPE member name

"string" is an integer or "M" (for many)

"name" is the data for import from, or export to, ADW/IEW

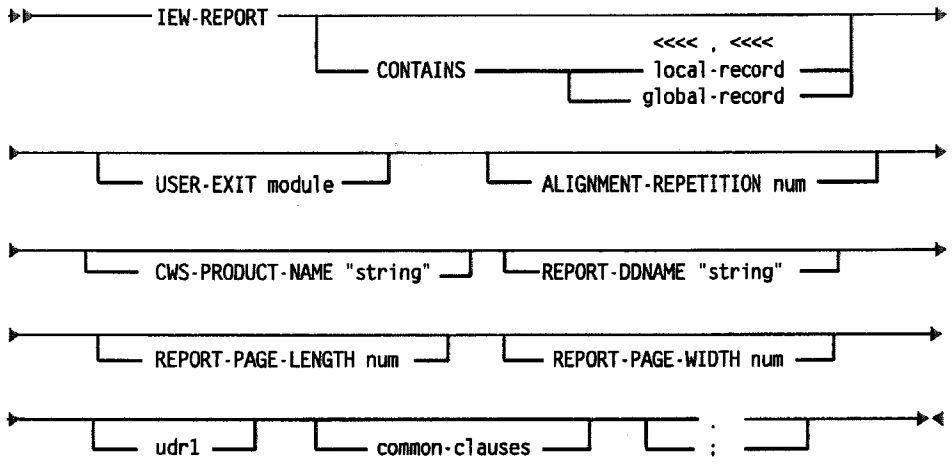
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-REPORT

ADW/IEW Design Workstation REPORT member type.

Syntax



where

local-record is an IEW-LOCAL-DATA-RECORD member name

global-record is an IEW-GLOBAL-DATA-RECORD member name

module is an IEW-MODULE member name

num is a positive integer

"string" is data to be imported from, or exported to, ADW/IEW

udr1 is defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

IEW-SCREEN

ADW/IEW Design Workstation SCREEN member type.

CONTAINS
Must contain the name of every SCREEN-OBJECT and DATA-STRUCTURE which is a part of this SCREEN.

KEYS
Initial cursor position on this screen.

SEE
Must contain the name of every GLOBAL-DATA-STRUCTURE or LOCAL-DATA-STRUCTURE this SCREEN is composed of.

USER-EXIT
Contains the name of a module by which this SCREEN is always processed, such as a validation module.

TP-MONITOR
The type of teleprocessing monitor.

SCREEN-WIDTH
Terminal column limit

SCREEN-DEPTH
Terminal row limit

INPUT-MAP-NAME
Input map set name

OUTPUT-MAP-NAME
Output Map set Name

BMS-INPUT-MAP-NAME
Name of the input map

BMS-OUTPUT-MAP-NAME
Output BMS map name

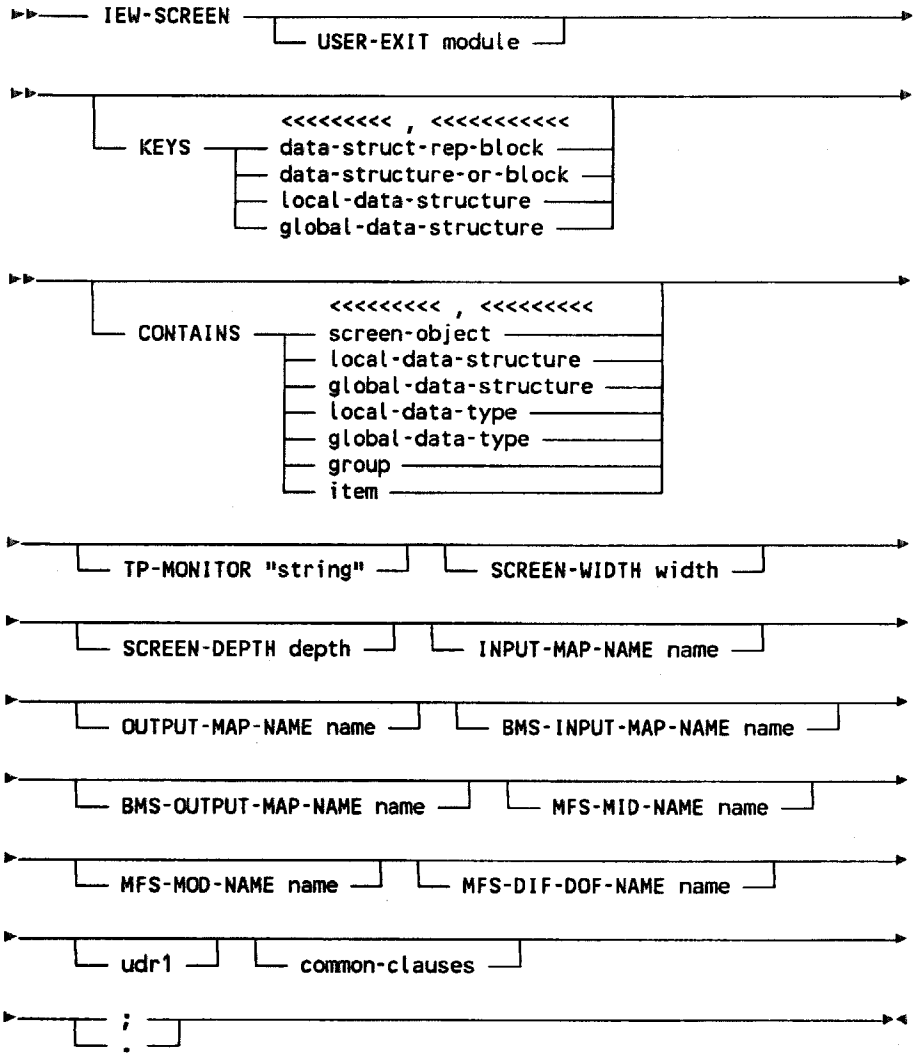
MFS-MID-NAME
MFS Input Map Name

MFS-MOD-NAME
MFS Output Map Name

MFS-DIF-DOF-NAME
Name of the MFS DIF/DOF

IEW-SCREEN

Syntax



where

module is an IEW-MODULE member name

global-data-structure is an IEW-GLOBAL-DATA-STRUCTURE member name

local-data-structure is an IEW-LOCAL-DATA-STRUCTURE member name

data-struct-rep-block is an IEW-DATA-STRUCT-REP-BLOCK member name

data-structure-or-block is an IEW-DATA-STRUCTURE-OR-BLOCK member name

screen-object is an IEW-SCREEN-OBJECT member name

global-data-type is an IEW-GLOBAL-DATA-TYPE member name

local-data-type is an IEW-LOCAL-DATA-TYPE member name

group is a GROUP member name

item is an ITEM member name

"string" is either "IMS/MFS" or "CICS/BMS"

width and depth are unsigned integers

name is the name of a repository member

udr1 is as defined in Appendix 2

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-SCREEN-OBJECT

ADW/IEW Design Workstation SCREEN-OBJECT member type.

MODIFIED
Modified Y or N

DISPLAYED
Is this screen object to be displayed?

SCREEN-ATTRIBUTE
Attribute for screen character.

DEPTH
DEPTH

ROW
ROW

COLUMN
COLUMN

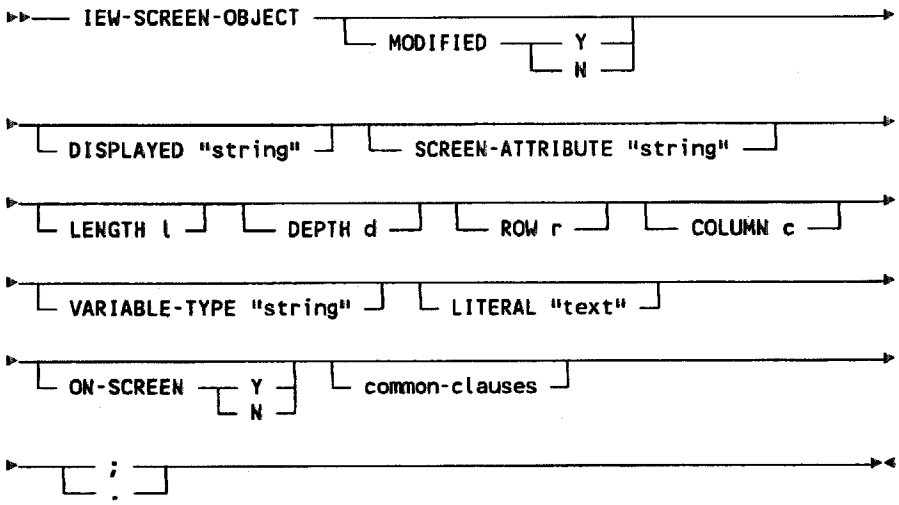
VARIABLE-TYPE
The type of variable

LENGTH
'Length'

LITERAL
Literal string

ON-SCREEN
On-screen indicator Y or N.

Syntax



where

"string" is the data for import from, or export to, ADW/IEW

d, r, c, v, and **l** are unsigned integers

"text" is up to **d** delimited strings, each string having a maximum length of **l** characters

common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-SEQUENTIAL-PROCESS

ADW/IEW Analysis Workstation SEQUENTIAL-PROCESS member type. An activity that is repeatedly executed within the enterprise, each execution of which produces a specific kind of effect on entities, or information about entities, of specific types.
A sequential process described in an action diagram consists of steps that are executed one at a time, in a prescribed order.

HAS
This clause contains the names of the SEQUENTIAL-PROCESSES that constitute this SEQUENTIAL-PROCESS.

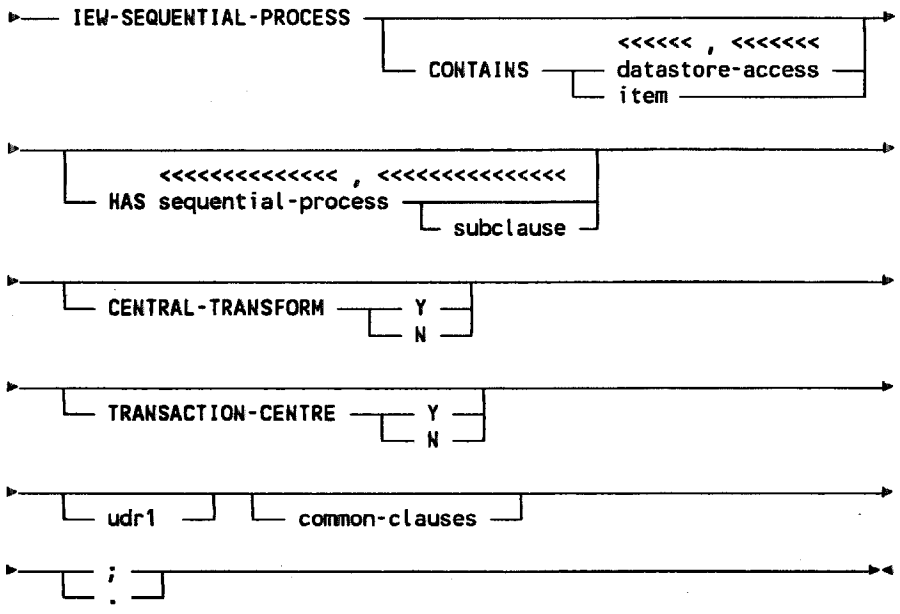
CONTAINS
Must contain the name of every DATASTORE-ACCESS which is a part of this SEQUENTIAL-PROCESS.

SEE
List of ENTITY-TYPES, ATTRIBUTE-TYPES or RELATIONSHIP-TYPES that are involved in this SEQUENTIAL-PROCESS.

CENTRAL-TRANSFORM
Is this object transformed centrally, Y or N.

TRANSACTION-CENTRE
Is this object a transaction centre, Y or N.

Syntax



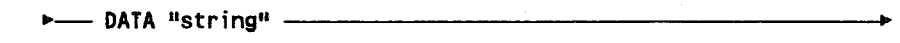
where

datastore-access is an IEW-DATASTORE-ACCESS member name

item is an ITEM member name

sequential-process is an IEW-SEQUENTIAL-PROCESS member name

subclause is



where "**string**" is the data for import from, or export to, ADW/IEW

udr1 is as defined in Appendix 2

common-clauses are any of the clauses available to all member types. Refer to Appendix 2 for details of the common clauses.

IEW-SUBJECT-AREA

ADW/IEW Planning Workstation SUBJECT-AREA member type.
A broad topic or category of information. Subject areas are mainly of use in discovering and grouping entity-types.
The group of entity types that pertain directly to a function or a major topic of interest to the enterprise.

CAUSES
Records the problems caused by this SUBJECT-AREA.

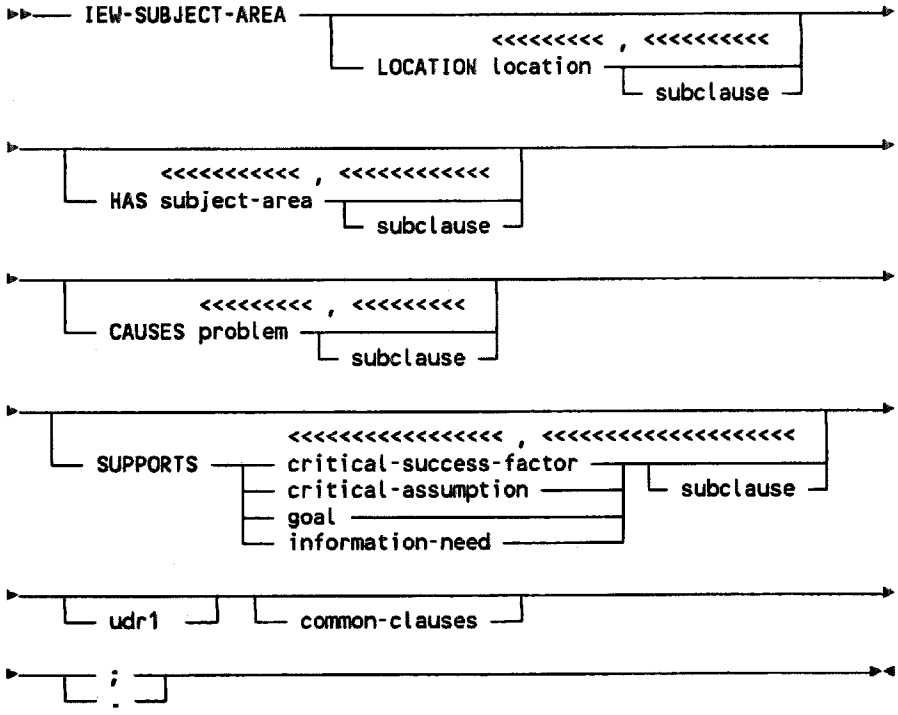
HAS
This clause contains the names of the SUBJECT-AREAs that constitute this SUBJECT-AREA.

SEE
List of ENTITY-TYPES, ATTRIBUTE-TYPES or RELATIONSHIP-TYPES that are involved in this SUBJECT-AREA.

LOCATION
States the location within the enterprise of this object by naming the appropriate LOCATION members.

SUPPORTS
Must contain the name of every planning object - GOAL, CRITICAL-ASSUMPTION, CRITICAL-SUCCESS-FACTOR, INFORMATION-NEED - which is part of this SUBJECT-AREA.

Syntax



where

location is an IEW-LOCATION member name

subclause is

where **"string"** is the data to be imported from, or exported to, ADW/IEW

subject-area is an IEW-SUBJECT-AREA member name

problem is an IEW-PROBLEM member name

critical-success-factor is an IEW-CRITICAL-SUCCESS-FACTOR member name

IEW-SUBJECT-AREA

critical-assumption is an IEW-CRITICAL-ASSUMPTION member name

goal is an IEW-GOAL member name

information-need is an IEW-INFORMATION-NEED member name

udr1 is as defined in Appendix 2

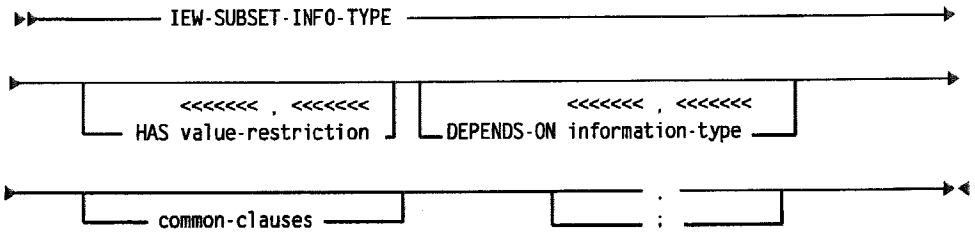
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-SUBSET-INFO-TYPE

ADW/IEW Analysis Workstation SUBSET-INFORMATION-TYPE member type.

Syntax



where

information-type is an IEW-INFORMATION-TYPE member name

value-restriction is an IEW-VALUE-RESTRICTION member name

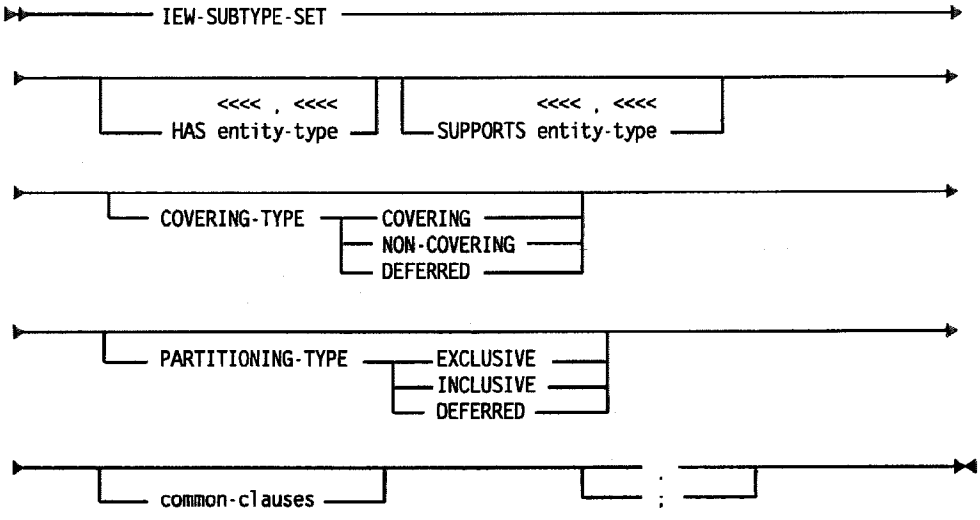
common-clauses are any of the clauses available to all member types

Refer to Appendix 2 for details of the common clauses.

IEW-SUBTYPE-SET

ADW/IEW Planning/Analysis Workstation SUBTYPE-SET member type.

Syntax



where

entity-type is an IEW-ENTITY-TYPE member name

common-clauses are any of the clauses available to all member types

Refer to Appendix 2 for details of the common clauses.

IEW-TBSPACE

ADW/IEW Design Workstation DB2-TABLESPACE member type. Tablespaces may be simple, that is not partitioned, or they may be partitioned.

If they are partitioned, then a corresponding cluster index must be defined in the dictionary as a DB2-INDEX member.

All the clauses available to define DB2-TBSPACE members are optional. However, the IN clause, defining the database to which the table space belongs, must be present for the successful generation of SQL CREATE TABLESPACE, DROP TABLESPACE or ALTER TABLESPACE statements.

In DB2, if a table space is partitioned, the corresponding index must also partitioned. Therefore for DB2-TBSPACE members defined with a PARTITION clause, you should define a corresponding clustered DB2-INDEX with the same partitions.

IN
To specify the database to which a table space belongs enter:

```
IN database
```

database is the name of a IEW-DB2-DATABASE member.

On encoding, the member specified in the IN clause is checked to ensure that it is a DB2-DATABASE member. The IN clause must be defined to successfully generated SQL CREATE TABLESPACE and CREATE TABLE statements. The DB2- DATABASE member named in the IN clause is used to generate a database-qualified name for the table space.

STOGROUP
To define the VSAM catalog the index, table space or partition is to use, enter:

```
VCAT-1 catalog      for the tablespace
VCAT catalog        for the partition
```

catalog is the name of a VSAM catalog, of no more than 8 characters.

You can additionally define further details of primary and secondary storage using the PRIQTY, SECQTY sub-clauses. Both sub-clauses are expressed in kilobytes.

To specify the amount of primary storage space, enter:

```
PRIQTY p
```

p is the number of kilobytes, and must be in the range 3 to 4194304 inclusive.

To specify the amount of secondary storage space, enter:

```
SECQTY s
```

s is the number of kilobytes, and must be in the range 0 to 131068 inclusive.

You can also specify whether or not DB2-defined data sets are to be erased when the tbspace or partition is deleted in a DROP command.

If you want the data sets to be erased, enter:

```
ERASE YES
```

If you do not want the data sets to be erased, enter:

```
ERASE NO
```

If you do not specify PRIQTY, SECQTY or ERASE sub-clauses, the DB2 defaults apply.

VCAT

To define a control or master level password used to access the VSAM catalog, enter:

```
PASSWORD password
```

password is a control or master level VSAM catalog password, of no more than eight characters. When you generate SQL statements, the VCAT password is taken directly from the password you specify in this clause.

FREEPAGE
You can accommodate future expansion of an index, table space or partition by defining the frequency with which pages are left free.

To define the relative frequency with which free pages are allocated, enter:

```
FREEPAGE-1          for the tablespace
FREEPAGE fn         for the partition
```

fn is an integer in the range 0 to 255. For example, FREEPAGE 4 means that 1 free page is left after every 4 pages.

If you do not define a FREEPAGE clause the DB2 defaults apply.

PCTFREE
You can accommodate future expansion of an index, table space or partition by defining the percentage of each page that is left free.

To define the percentage space kept free on a page, when an index, table space or partition is loaded or reorganized, enter:

```
PCTFREE-1 pn        for the tablespace
PCTFREE pn          for the partition
```

pn is an integer in the range 0 to 99. If you do not define a PCTFREE clause the DB2 defaults apply.

PARTITION
To define that an index or table space is to be partitioned, enter:

```
PARTITION
```

You can optionally define a number, details of storage, free space allocation and key values, for each partition. To successfully generate SQL CREATE INDEX statements from DB2-INDEX members containing a PARTITION clause, you must define a NUMBER and KEY clause for each PARTITION. To successfully generate SQL CREATE TABLESPACE statements from DB2-TBSPACE members containing a PARTITION clause, you must define a NUMBER clause for each PARTITION.

To give the partition a number, enter:

```
NUMBER n
```

n is an integer in the range 1 to 64. If you do not define the numbers of partitions, they are automatically generated by the DB2 CREATE command, commencing with 1, and increasing in increments of 1.

To define that the partitions of an index have a key value, enter:

```
KEY 'key-val'
```

key-val is the highest value, within delimiters, that a column in the partition can have.

If you are indexing more than one column, each key value must be enclosed in quotes and separated by a comma. The first key value corresponds to the first index column, the second value to the second index column and so on.

If the key value is a character field it must be enclosed in single quotes within double quotes. For example:

```
KEY " 'key-val' "
```

Note: you cannot define a key clause in DB2-TB-SPACE members. To define the storage space to which the partition belongs, use the VCAT, or STOGROUP, PRIQTY, SECQTY and ERASE clauses. To define how much space is left free in the partition, use the FREEPAGE and PCTFREE clauses.

If you do not define storage or free space for partitions individually, they automatically take space from that specified for the index or table space as a whole.

BUFFERPOOL
To define the buffer pool that table spaces use, enter:

```
BUFFERPOOL bufferpool-name
```

bufferpool-name is one of the following buffer pools:

```
BP0          BP1
BP2          BP32K
```

The buffer pool defined in the DB2-DATABASE definition is the default used by those table spaces that belong to the database, and do not have a buffer pool specified in their own member definition.

LOCKSIZE
To define the size of the storage unit that can be locked, enter:

```
LOCKSIZE unit
```

unit is one of the following units of locking:

```
ANY                PAGE
TABLESPACE         TABLE
```

If you do not specify a LOCKSIZE the DB2 default applies.

CLOSE
To define that the data set, on which an index or table space resides, is to be closed when not in use, enter:

```
CLOSE YES
```

To define that it is to remain open, enter:

```
CLOSE NO
```

If you do not define a CLOSE clause the DB2 default applies.

DSETPASS
To define a password for the VSAM data set, on which an index or table space resides, enter:

```
DSETPASS password
```

password is a VSAM data set password of no more than 8 characters.

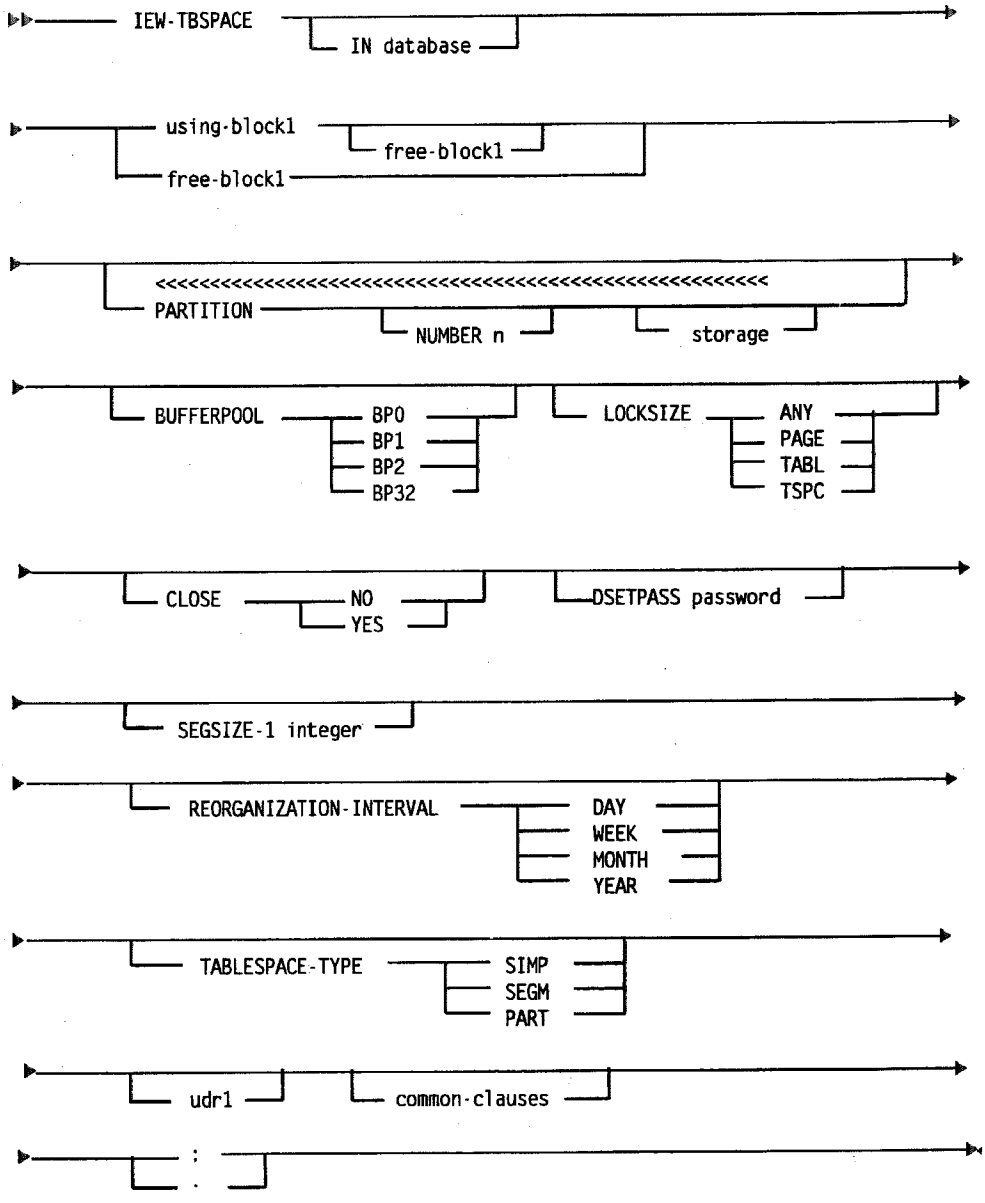
IEW-TBSPACE

SEGSIZE-1
To define a segmented table space and the number of pages to be allocated to each segment, enter:

```
SEGSIZE-1 integer
```

integer is a multiple of 4, in the range 4 to 64 inclusive.

Syntax

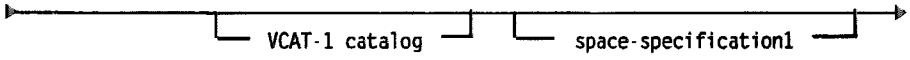


where

IEW-TBSPACE

database is an IEW-RELATIONAL-DATABASE member name

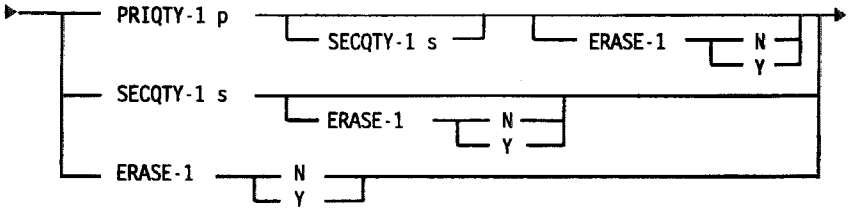
using-block1 is



where

catalog is a VSAM catalog name of no more than 8 characters

space-specification1 is

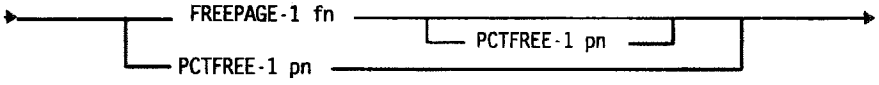


where

p is an integer in the range 3 to 419304

s is an integer in the range 0 to 131068

free-block1 is



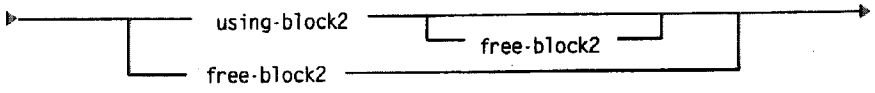
where

fn is an integer in the range 0 to 255

pn is an integer in the range 0 to 99

n is an integer in the range 1 to 64

storage is



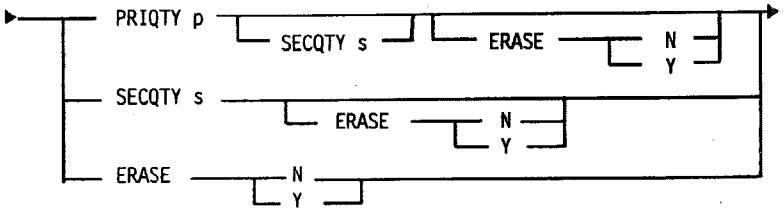
where

using-block2 is

where

catalog is as defined above

space-specification2 is



where

p is as defined above

s is as defined above

free-block2 is

where

fn is as defined above

pn is as defined above

password is a VSAM data set password, of no more than 8 characters

udr1 is defined in Appendix 2

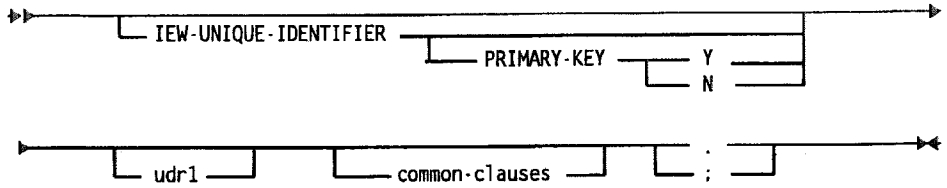
common-clauses are any of the common clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-UNIQUE-IDENTIFIER

ADW/IEW Design Workstation UNIQUE-IDENTIFIER member type.

Syntax



where

udr1 is defined in Appendix 2

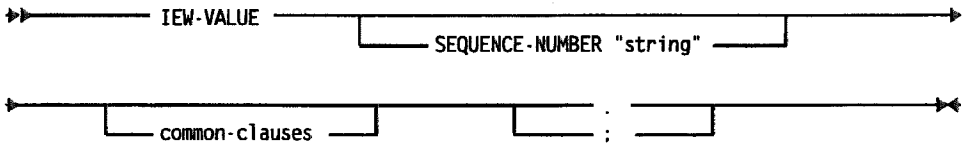
common-clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

IEW-VALUE

ADW/IEW Analysis Workstation VALUE member type.

Syntax



where

"string" is data to be imported from, or exported to, ADW/IEW

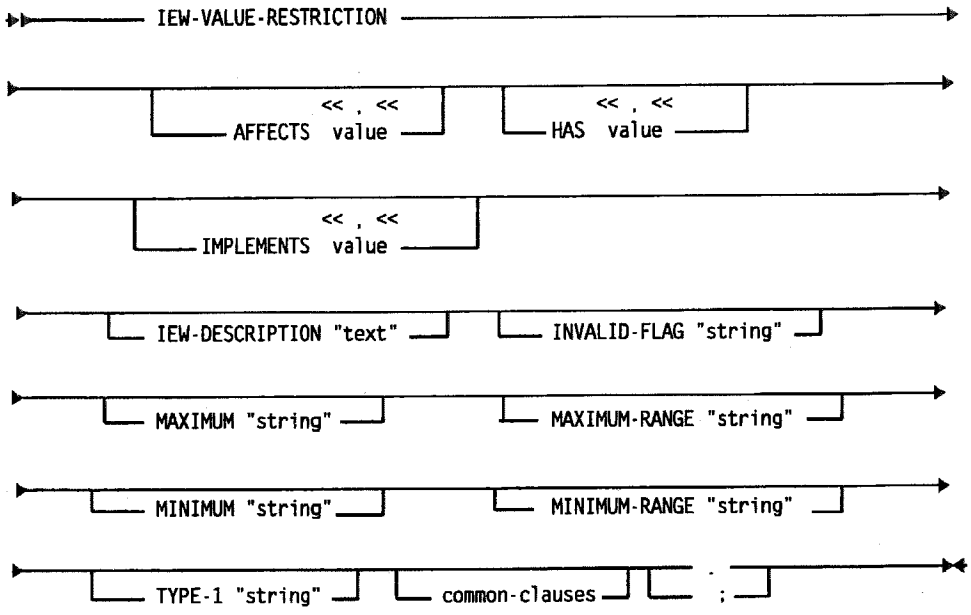
common-clauses are any of the clauses available to all member types

Refer to Appendix 2 for details of the common clauses.

IEW-VALUE-RESTRICTION

ADW/IEW Analysis Workstation VALUE-RESTRICTION member type.

Syntax



where

value is an IEW-VALUE member name

"text" is up to 32767 delimited strings, each string having a maximum length of 60 characters

"string" is data to be imported from, or exported to, ADW/IEW

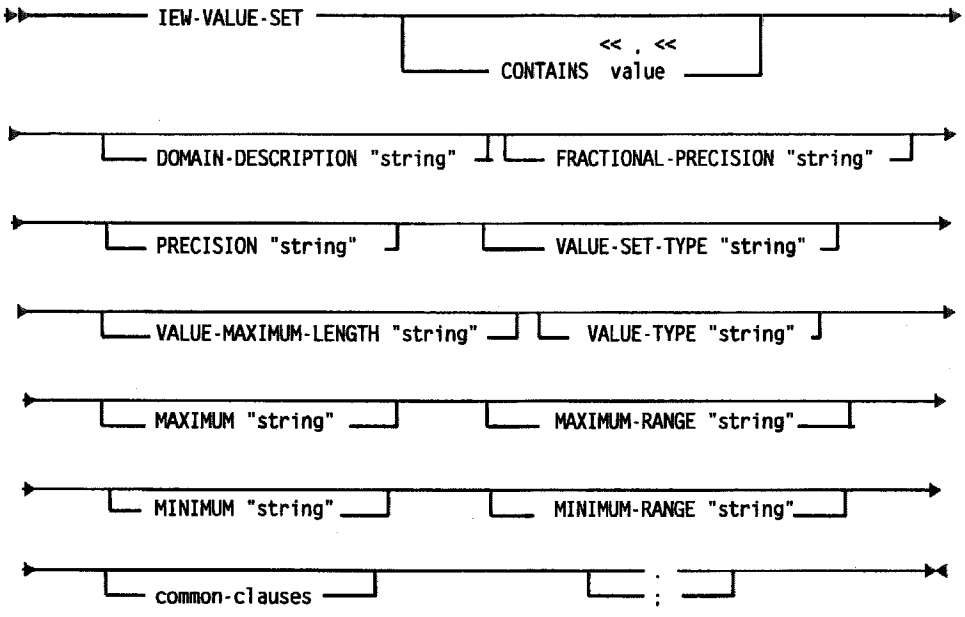
common-clauses are any of the clauses available to all member types

Refer to Appendix 2 for details of the common clauses.

IEW-VALUE-SET

ADW/IEW Analysis Workstation VALUE-SET member-type.

Syntax



where

value is an IEW-VALUE member name

"string" is data to be imported from, or exported to, ADW/IEW

common-clauses are any of the clauses available to all member types

Refer to Appendix 2 for details of the common clauses.

This page is intentionally blank.

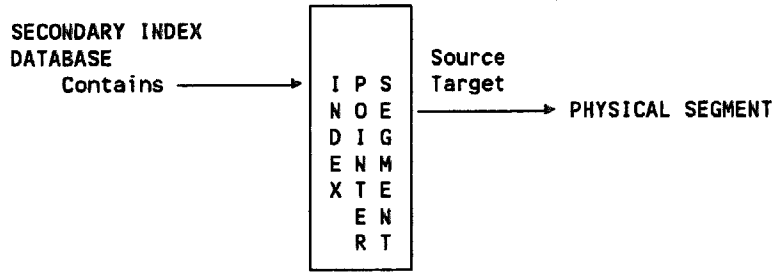
IMS

IMS

See entries under **DL1**.

INDEX-POINTER-SEGMENT

The INDEX POINTER SEGMENT records information relevant to the secondary indexing of IMS databases.
The more commonly used relationships to and from INDEX POINTER SEGMENT are:



For further details refer to the "IMS (DL/1) Interface" documentation.

ATTRIBUTES CONTAINS
This clause should include all the contained ITEMs or GROUPs (which may be ALIGNED by adding this keyword after the list entry)

FREQUENCY
The number of segments held per parent; or an integer for the number of root segments (for child segments there may be two decimal points).

SEQUENCE-KEY
Specifies the IMS name for the sequence key.
An internal member will be generated with this name which has "uses" links with the XDFLD and each of the SUBSEQ fields.

GENERATES
Details of all items for which it is required to generate FIELD statements.

RELATED-TO
Name the target physical segment for this index pointer

ON
Names the index search field as referenced in SSA's.

INDEX-POINTER-SEGMENT

POINTERS
Specifies whether DIRECT (RBA) pointers are used for the index, or SYMBOLIC (key value) pointers should be used.
DIRECT is more performant, SYMBOLIC less sensitive to reorganisation.

SOURCE
Names the source segment for this index to be built from.

CONSTANT
Constant byte for shared indices (e.g. CHARACTER 'x', HEXADECIMAL 'hh', BITS 'bbbbbbbb')

SEARCH-KEY-FIELDS
List the search key fields for this index pointer segment.
This entry is required.

SUBSEQUENCE-FIELDS
No help text defined

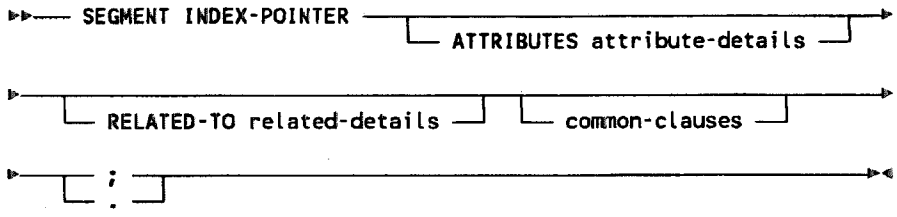
DUPLICATE-DATA-FIELDS
List duplicate (user) data fields required in index segment, up to five off.

SUPPRESSING-ON
No help text defined

MAINTENANCE-EXIT
Name of the secondary index maintenance exit (if applicable).

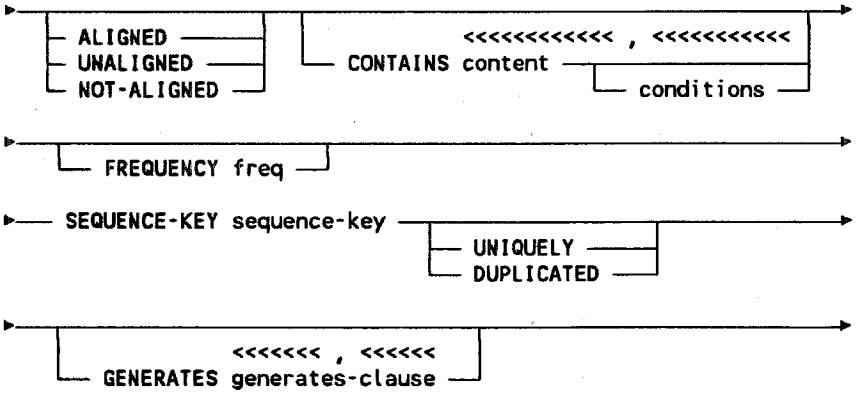
CONCATENATED-KEY-NAME
Enter the IMS name for the concatenated key field

Syntax



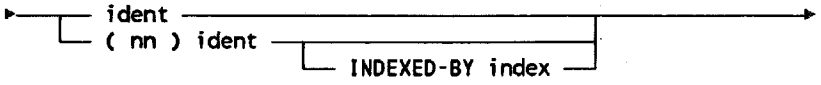
where:

attribute-details are:

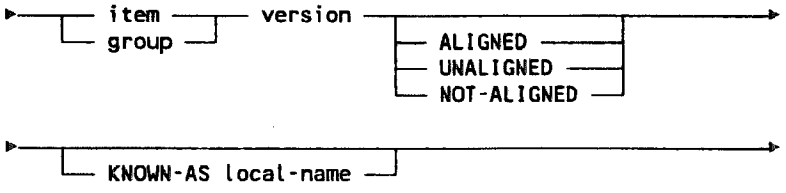


where

content is:



where **ident** is:



where:

item is the name of an ITEM repository member

group is the name of a GROUP repository member

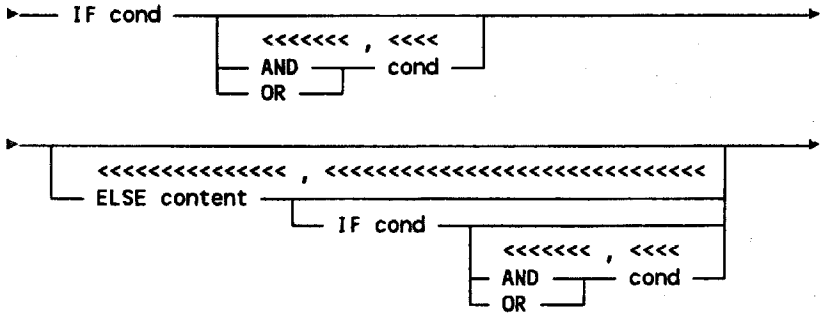
version is an unsigned integer in the range 1 to 15

local-name is a name, conforming to the rules for member names stated in the CONTROLMANAGER User's Guide (CMR-UG)

nn is an unsigned integer of from 1 to 18 digits, being the number of times **item** or **group** occurs in the array

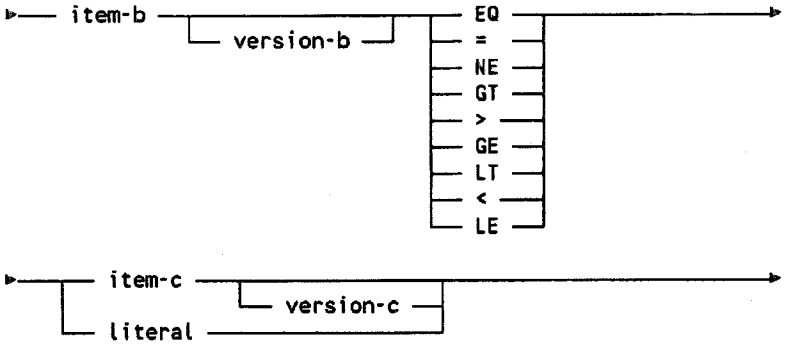
index is a name, conforming to the rules for member names, that is to be used as the index name when COBOL data descriptions are generated by the Source Language Generation facility

conditions are:



where

cond is:



where:

literal is a literal comparand

item-b is the name of the ITEM whose contents are to be compared with the comparand

version-b is an unsigned integer in the range 1 to 15

item-c is the name of the ITEM whose contents are the comparand

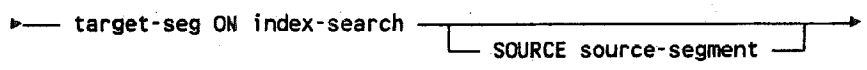
version-c is an unsigned integer in the range 1 to 15

content is as defined above

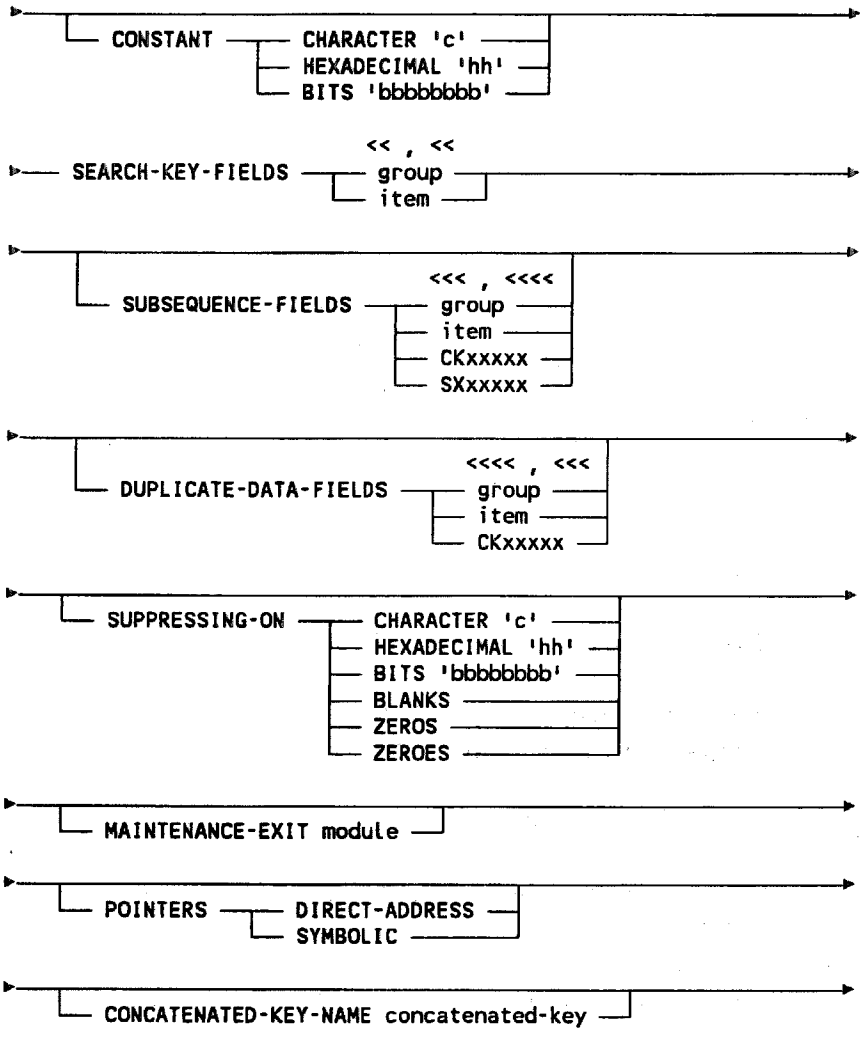
freq is an unsigned integer in the range 1 to 16777215

sequence-key is a 1 to 8 character unique alphanumeric name

related-details are:



INDEX-POINTER-SEGMENT



where

target-seg is the name of a SEGMENT that is a PHYSICAL TARGET-SEGMENT member

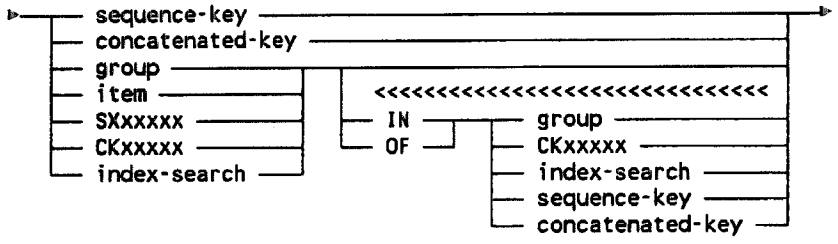
index-search-field is a 1 to 8 character unique alphanumeric name

source-segment is the name of a SEGMENT that is a PHYSICAL SOURCE-SEGMENT member

module is the name of a MODULE member

group and **item** are as defined above

generates-clause is:



where

CKxxxxx is a 3 to 7 character name to be applied to a system related field within a GENERATES, SUBSEQUENCE-FIELDS or DUPLICATE-DATA-FIELDS clause. The name must be unique and start with CK.

SXxxxxx is a 3 to 7 character name to be applied to a system related field within a GENERATES or SUBSEQUENCE-FIELDS clause. The name must be unique and start with SX.

concatenated-key is used when a logical child segment is being defined, to specify the name to be given to the destination parent's concatenated key.

c is any printable character

hh is a hexadecimal representation of any printable or non-printable character

bbbbbbbb is a one byte bit string representation of any printable or non-printable character

common clauses are any of the clauses available to all member types.

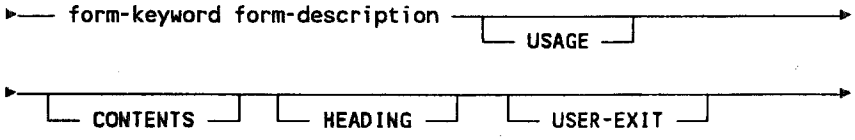
Refer to Appendix 2 for details of the common clauses.

ITEM

An ITEM is a fundamental element of data, the smallest named unit into which data is divided in the organization.

An ITEM member consists of:

where **form** is:



An ITEM member definition must commence with the member identifier keyword ITEM. The clauses and keywords forming the body of the member definition statement are all optional. They can be specified in any order except for **form-keyword** which if used, can only be preceded by common clauses and must be immediately followed by an associated **form-description.** Each form-keyword may optionally have a USAGE, CONTENTS, HEADINGS and USER-EXIT clause. Up to 15 versions of the form-keyword can be defined in an ITEM member.

Any common clauses must be declared either **before** the form keyword or **after** that form keyword's associated form-description and clauses.

Refer to page 2-429 for the syntax of the ITEM member type.

ENTERED-AS, HELD-AS, REPORTED-AS, DEFAULTED-AS: The Form-Keyword

Form-keyword is one of the following:

ENTERED-AS	the form in which the item is input
HELD-AS	the form in which the item is held and processed within the computer, and/or held in its secondary storage

REPORTED-AS	the form in which the item is output
DEFAULTED-AS	when the item may relate to either input internal processing or output

The form-keyword may be followed by a version number and must be followed by a form-description.

An ITEM member definition can contain a maximum of 15 form-keywords. Within this total up to 15 versions of each of the keywords ENTERED-AS, HELD-AS and REPORTED-AS can be specified in any combination. Only one DEFAULTED-AS keyword is allowed.

Where two or more versions of a form is specified then each must be identified with a version number which need not be consecutive but must be in the range of 1 to 15. If a version number is not specified then a default value of 1 is assumed.

For example:

```
ITEM
DEFAULTED-AS    form-description
ENTERED-AS      form-description
HELD-AS 1       form-description
HELD-AS 2       form-description
REPORTED-AS     form-description
;
```

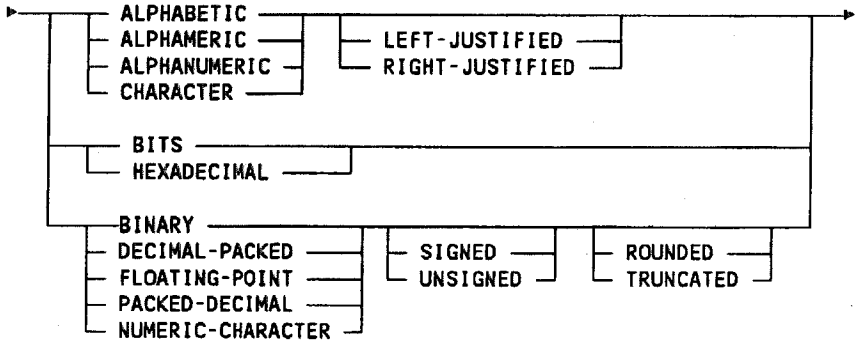
Here there are two versions of the HELD-AS (internal processing) form. Since the version numbers for ENTERED-AS and REPORTED-AS are omitted, both are defaulted to one.

The Form-Description

A **form-description** directly follows a form-keyword, and defines the nature and form of the data described by the ITEM.

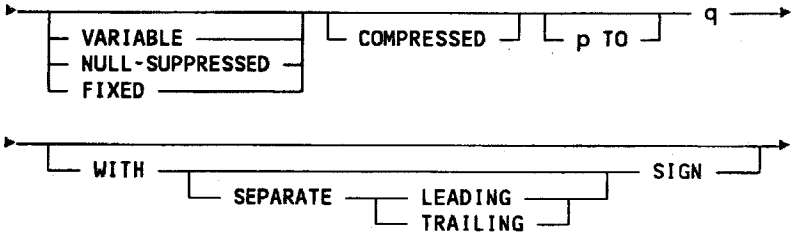
ITEM

The first part of form-description can be written as follows:



The keywords SIGNED and UNSIGNED may alternatively precede their associated form-description keywords except where that keyword is FLOATING-POINT.

This must be followed by a length as follows:



p and **q** are unsigned decimal numbers specifying respectively the minimum and maximum lengths of the item. The elements **p TO** are present only in the definition of a variable length item. The element **q** may be zero. If it is zero, MANAGER Products will assume a value of one. The maximum value that can be declared for the length **q** depends on the associated form-description keyword.

Certain form-description keywords are relevant to Source Language Generation, though they can be used in other contexts.

The ROUNDED and TRUNCATED keywords are relevant to MARK IV Source Language Generation.

The VARIABLE, NULL-SUPPRESSED and FIXED keywords are relevant to ADABAS Source Language Generation. The COMPRESSED keyword is relevant to SYSTEM 2000/80 Source Language Generation.

The WITH clause may be used only when both NUMERIC-CHARACTER is specified and UNSIGNED is not specified.

If the keyword SEPARATE is included in the WITH clause, then when Source Language Generation is performed from the member, an extra byte is generated to contain the sign. If SEPARATE is omitted from the WITH clause, the clause specifies which end of the field defined by the member contains the sign overpunch.

One way of describing the item is explained here. Alternatively, you can describe the format of a data item by means of a PICTURE clause, or you can describe the item as having the same format as another item by using a NAME clause.

Form-Description Keywords

If the **ALPHABETIC** form-description keyword is specified:

- the item can contain letters of the alphabet and/or spaces
- the maximum value that can be declared for the length is 32767
- the units to which the length relates are characters.

If the **ALPHAMERIC** or **ALPHANUMERIC** form-description keyword is specified:

- the item can contain letters of the alphabet and/or numeric characters (hexadecimal F0 to F9) and/or spaces (hexadecimal 40)
- the maximum value that can be declared for the length is 32767
- the units to which the length relates are characters.

If the **BINARY** form-description keyword is specified:

- the item can contain a binary number.
- the maximum value that can be declared for the length is 18 or 18.0

- the units to which the length relates are decimal digits in the decimal equivalent of the maximum number the item can hold.

If the **BITS** form-description keyword is specified:

- the item can contain a bit string.
- the maximum value that can be declared for the length is 4096
- the units to which the length relates are bits.

If the **CHARACTER** form-description keyword is specified:

- the item can contain any printable and/or non-printable characters
- the maximum value that can be declared for the length is 32767
- the units to which the length relates are characters.

If the **DECIMAL-PACKED** or **PACKED-DECIMAL** form-description keyword is specified:

- the item can contain a packed decimal number (relevant to COBOL COMPUTATIONAL-3 or PL/I DECIMAL FIXED)
- the maximum value that can be declared for the length is 18 or 18.0
- the units to which the length relates are decimal digits.

If the **FLOATING-POINT** form-description keyword is specified:

- the item can contain exponent and mantissa
- the maximum value that can be declared for the length is 16
- the units to which the length relates are decimal digits in the mantissa.

If the **HEXADECIMAL** form-description keyword is specified:

- the item can contain a hexadecimal string
- the maximum value that can be declared for the length is 512
- the units to which the length relates are hexadecimal characters.

If the NUMERIC-CHARACTER form-description keyword is specified:

- the item can contain numeric characters (hexadecimal F0 to F9)
- the maximum value that can be declared for the length is 18 or 18.0
- the units to which the length relates are decimal digits.

The maximum and minimum length of an item with a BINARY, DECIMAL-PACKED, NUMERIC-CHARACTER or PACKED-DECIMAL form-description is expressed in the form:

```
n.m
```

n is the number of digits before the decimal point
m is the number of digits after the decimal point. **n** plus **m** must not exceed 18.

Any non-significant zeros either before or after the decimal point are omitted. Therefore, if n is zero n can be omitted and if m is zero m can be omitted. For example, when m is zero, a value of 18.0 is expressed as 18. The decimal point itself is omitted if the figure is an integer.

PICTURE: Specifying the Form-Description Pictorially

To represent the form-description of an ITEM pictorially with symbols enter:

```
PICTURE 'picture'
```

picture is a string of one to thirty characters. The character string must contain only picture symbols and repetition factors. It must contain at least:

- one of the symbols **A X Y Z 9 ***, or

- a floating **S £ +** or **-** (that is, a leading string of two or more consecutive occurrences of one of these symbols).

Repetition Factor

A repetition factor is an integer in parentheses which specifies the number of consecutive occurrences of the symbol it immediately follows. For example:

A(5)

is equivalent to AAAAA.

The value of the repetition factor must be in the range 1 to 32767.

Calculating the Length of an Item

The length of an item in reports is the sum of the repetition factors in the picture, subject to the following conditions:

- if the repetition factor of a symbol is not stated, then it is assumed to have a value of one

- the implied repetition factor of any **K P** or **V** symbol is excluded from the count

- in a numeric floating point picture, if the sign symbol of the mantissa and/or the exponent is omitted,
  then the defaulted sign symbol is included in the count.

Calculating the Number of Digit Positions

In numeric pictures the number of digit positions is the sum of the repetition factors of the following symbol:

9

In numeric edited pictures the number of digit positions is the sum of the repetition factors of the following symbols:

I R T Y Z 0 9 *

plus for each of the following symbols, the total of its repetition factors minus one:

S £ + -

In numeric floating point pictures the number of digit positions is the sum of the repetition factors of the following symbols:

I R T 9

PICTURE Clause Symbols

Picture symbol	Description
A	Any letter or a space.
B	A space is inserted unless the position is required for a floating symbol.
CR	The characters CR are inserted if the item is negative. Otherwise, spaces are inserted.
DB	The characters DB are inserted if the item is negative. Otherwise, spaces are inserted.
E	Inserted in floating-point pictures on output to indicate that the exponent follows.
I	A decimal digit which is overpunched in the 12 punching position, signifying + , if the item is positive or zero and is output to a card punch.
K	The position of an implied exponent symbol in a floating-point PICTURE clause.
P	Implied decimal scaling position, if the location of the decimal point falls outside the item.
R	A decimal digit which is overpunched in the 11 punching position, signifying - , if the item is negative and is output to a card punch.

ITEM

Picture symbol	Description
S	A + symbol is inserted if the item is positive or zero, a - symbol if the item is negative; or a floating sign position.
T	A decimal digit which, if the item is output to a card punch, is overpunched in the 12 punching position, signifying + , if the item is positive or zero, or in the 11 punching position, signifying - , if the item is negative.
V	The position of an implied decimal point.
X	Any character.
Y	A zero-suppressed decimal digit (replaced by a space if zero).
Z	A leading zero-suppressed decimal digit replaced by a space if zero and not preceded by significant digits.
/	A position into which the character / is inserted.
+	In numeric edited pictures: - a + symbol is inserted if the item is positive or zero - a space is inserted if the item is negative - a floating + position. In numeric floating point pictures: - a + symbol is inserted if the following value (mantissa or exponent) is positive or zero - a space is inserted if the following value (mantissa or exponent) is negative.

Picture symbol	Description
-	In numeric edited pictures: - a - symbol is inserted if the item is negative - a space is inserted if the item is positive or zero - a floating - position. In numeric floating point pictures: - a - symbol is inserted if the following value (mantissa or exponent) is negative - a space is inserted if the following value (mantissa or exponent) is positive.
,	A position into which a comma is inserted providing there is a non-suppressed preceding digit.
*	A decimal digit to be replaced by the symbol * if it is zero and not preceded by significant digits.
£	A symbol is inserted, or a floating position.
	A decimal point is inserted on output.
0	A position into which a zero is inserted on output unless the position is needed for a floating symbol or rendered to a space by a floating symbol.
9	A decimal digit.

Note that £ can be replaced by any local currency character whose internal code is hexadecimal 5B.

The valid picture symbols in each picture type are summarized below.

PICTURE TYPE	FORMAT SYMBOL:																					
	A	B	CR	DB	E	I	K	P	R	S	T	V	X	Y	Z	0	9	£	+	–	*	,	.	/
ALPHABETIC	*																							
ALPHANUMERIC	*												*				*							
ALPHANUMERIC EDITED	*	*											*			*	*							*
NUMERIC								*				*					*							
NUMERIC EDITED		*	*	*		*		*	*	*	*	*		*	*	*	*	*	*	*	*	*	*	*
NUMERIC FLOATING POINT					*	*	*		*		*	*					*		*	*			*	

PICTURE Clause Floating Symbols

The following symbols may be floating symbols:

```
S + - £
```

The above symbols become floating symbols if they occur in a picture as two or more consecutive leading characters, or as three or more leading characters interspersed, but not before the second character position, with any combination of the following:

- one full stop or one V symbol
- any number of B symbols
- any number of 0 symbols
- any number of comma symbols

The string of characters from the first to the last floating symbol together with any of the insertion characters **B 0** or **comma** immediately following the last floating symbol, is called a floating string.

If the floating string does not contain a full stop the second and subsequent occurrences of the floating symbol represent numeric digit positions which, together with any preceding interspersed symbols, are not preceded by a significant digit. The floating symbol is inserted into the character position immediately preceding the first digit position occupied by a significant digit.

For example, if a picture is declared as and the item contains zero, the item is output as all spaces, while if a picture is declared as 9 and the item contains zero, the item is output as 0 (because the symbol 9 implies that the digit is to be regarded as significant regardless of its value). If the item contains one, and the picture is or 9 , it is output as 1.

If the floating string contains a full stop the same rules apply, except that if the item is non-zero, the floating symbol can float only as far as the character position immediately preceding the full stop (decimal point).

Alphabetic PICTURE Clause

The permitted picture symbol is:

A

The maximum number of character positions is 32767.

The following are valid alphabetic PICTURE clauses:

- 'A'
- 'A(1)'
- 'A(32767)'
- 'AA(32765)A'.

Note that the second example is valid, although the repetition factor of one is superfluous.

The following are invalid alphabetic PICTURE clauses:

- 'A(32768)'
- 'AA(32766)A'
- 'AAA(23)AA(100)AAAAAAAAAAAAAAAAA'.

The reasons for invalidity are respectively:

- the repetition factor is greater than 32767

- the length of the item would be greater than 32767, that is, 1 (A) + 32766 (32766 times A) + 1 (A) = 32768

- the character string has more than 30 characters.

Alphanumeric PICTURE Clause

The permitted picture symbols are:

```
A X 9
```

The maximum number of character positions is 32767.

The following are **valid** alphanumeric PICTURE clauses:

- 'X'
- 'X9'
- 'AX'
- '9XA'
- 'A9'
- 'A(32765)X9'.

The following is an **invalid** alphanumeric PICTURE clause:

- 'A(32766)X9'.

The example is invalid because the length of the item would be greater than 32767, that is, 32766 (32766 times A) + 1 (X) + 1 (9) = 32768.

Alphanumeric Edited PICTURE Clause

The permitted picture symbols are:

```
A B X 0 9
```

The maximum number of character positions is 32767.

An alphanumeric edited PICTURE clause must contain:

- at least one B and one X, or
- at least one 0 and one X, or
- at least one 0 and one A.

The following are valid alphanumeric edited PICTURE clauses:

- 'XXXBXXX'
- 'XX00XXBAA'
- 'A0'.

The following are invalid alphanumeric edited PICTURE clauses:

- 'AB'
- 'B0'.

Numeric PICTURE Clause

The permitted picture symbols are:

P V 9

The maximum number of character positions is 18 for digits and 1 for sign.

The rules governing numeric PICTURE clauses are as follows:

- the symbol P may only occur as a string of characters to the left or right of a 9 symbol

- if there are P symbols, then V may only occur:

  - to the left of a left-hand P string, or
  - to the right of a right-hand P string.

- if there are no P symbols then a V symbol can occur anywhere.

The following are valid numeric PICTURE clauses:

- '999'
- '99PPP'

- '999V99'
- '99PPV'
- 'PP999'
- '9(8)V9(10)'
- 'VPP99'.

The following are **invalid** numeric PICTURE clauses:

- 'PP999PP'
- 'PP99V'
- 'V99PP'
- '9(8)V9(11)'.

The reasons for invalidity are respectively:

- the P symbols occur on both sides
- the V symbol is misplaced
- the V symbol is misplaced
- the length of the item would be greater than 19, that is, 8 (8 times 9) + 1 (V) + 11 (11 times 9) = 20.

Numeric Edited PICTURE Clause

The permitted picture symbols are:

```
B CR DB I P R ST V Y Z 0 9 £ * / + - comma and full stop
```

The maximum number of character positions is 127, of which the maximum number of digit positions is 18.

The rules governing numeric edited PICTURE clauses are as follows:

- the following symbols must not occur more than once: CR DB I R T V and full stop

- the symbols in the following groups are mutually exclusive:

  - P and full stop
  - + - S CR DB T I and R
  - Z * and floating + - S.

- the symbols S + and - can occur only as the first or last character of the picture, or in a floating string. It must not be both the first and the last character unless it is in a floating string extending over the whole picture

- the only characters that can precede a symbol are a single S + - or characters of a floating string

- the last symbol must not be / or full stop or comma

- an * or Z symbol must not occur to the right of the symbols 9 T I R, nor to the right of a full stop or V symbol if a 9 symbol is to the right of the * or Z symbol

- if there are two or more P symbols, they must be consecutive and either:

  - to the left of the picture, optionally preceded by a single S + - or symbol (or by a V symbol which though permitted is redundant), or

  - to the right of the picture, optionally followed by a single S + - CR or DB symbol (or a V symbol which though permitted is redundant)

- CR or DB can occur only as the last symbol.

The following are **valid** numeric edited PICTURE clauses:

- ''
- 'PPP99+'
- '++V+'
- '999PPV'
- '---B--'
- 'ZZ99'
- ',,9.99'
- 'R9(5)V9'.

The following are **invalid** numeric edited PICTURE clauses:

- '++VV+'
- 'CR999'
- '++SSS'

- '99/99/99/'
- 'VSP99'
- '999ZZ'
- '999'
- '9PP99'.

Numeric Floating Point PICTURE Clause

The permitted picture symbols are:

```
E I K R T V 9 + - and full stop
```

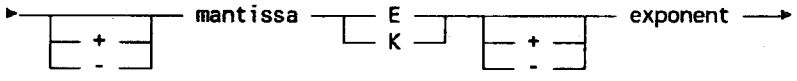
The maximum number of character positions is:

- mantissa 17 (16 for digits and 1 for sign)
- exponent 3 (2 for digits and 1 for sign)
- inserted E 1.

The rules governing numeric floating point PICTURE clauses are:

- the range is 5.4 x 10 to exponent -79 to 0.72 x 10 to exponent +76

- the picture is expressed as:



mantissa can contain:

- from one to sixteen 9 symbols, or
- from one to fifteen 9 symbols and one I R or T symbol, or
- one I R or T symbol, and optionally
- a full stop and/or a V symbol.

The symbols can be in any order. If an I R or T is included, there must be no preceding + or - symbol.

exponent can contain:

- one or two 9 symbols or
- a 9 symbol with one I R or T symbol or
- one I R or T symbol

If an I R or T symbol is included, there must be no preceding + or - symbol.

- if you omit the sign symbol from before the mantissa or the exponent, and no I R or T symbol occurs in either of them, a - symbol is defaulted. You will see a warning message and processing will continue.

The following are **valid** numeric floating point clauses:

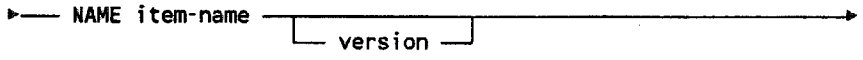
- '99V9E-99'
- '99.9E-9'
- '+V9(16)E+9'
- '9(7)K9'
- '-.9K+99'.

The following are **invalid** numeric floating point clauses:

- '+9(29)E+99'
- '9.9K999'
- 'E+99'
- '9.9E7'
- '9.99V9E99'
- '.E-99'.

NAME: Specifying That Two Items Have the Same Form-Description

To show that two items in a dictionary have the same form-description, enter:



item-name is the name of a referenced ITEM which has the same form-description as the defined ITEM

version identifies a particular version of the referenced ITEM. If you do not specify a number, version 1 is assumed.

Use of the NAME form-description has three effects:

- the defined ITEM will have the same form-description as the referenced ITEM

- if the referenced ITEM has a USAGE clause and the defined ITEM has not, then this USAGE clause will also apply to the defined ITEM. However, if both the referenced ITEM and the defined ITEM have USAGE clauses, then the USAGE clause for the defined ITEM will override that of the referenced ITEM. The same is true for the CONTENTS, HEADINGS, and USER-EXIT clauses.

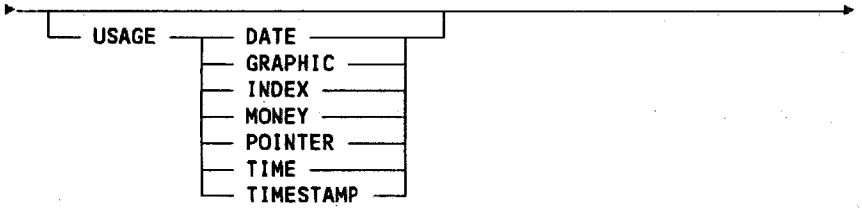
- any change in the form-description of the referenced ITEM will automatically change that of the defined ITEM. The same will apply to any shared USAGE, CONTENTS, HEADINGS and USER-EXIT clauses.

The form-description of an ITEM must be compatible with its CONTENTS clause. This applies to CONTENTS clauses shared with a referenced ITEM as well as those associated solely with the ITEM being defined.

USAGE Clause

The optional USAGE clause is mainly used for Source Language Generation, but may also be used for documentation purposes.

The clause has the form:



The DATE and MONEY keywords are particularly relevant to System 2000/80 Source Language Generation.

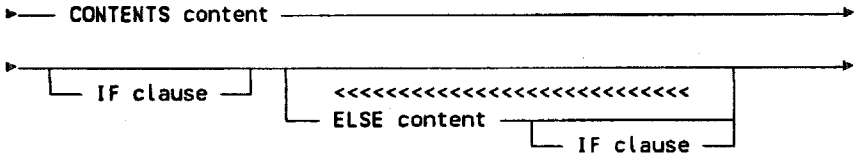
The POINTER keyword is particularly relevant to the generation of PL/I data description statements; an item with this usage can be generated as a PL/I pointer variable.

The INDEX keyword requires the item to have a minimum length p of four.

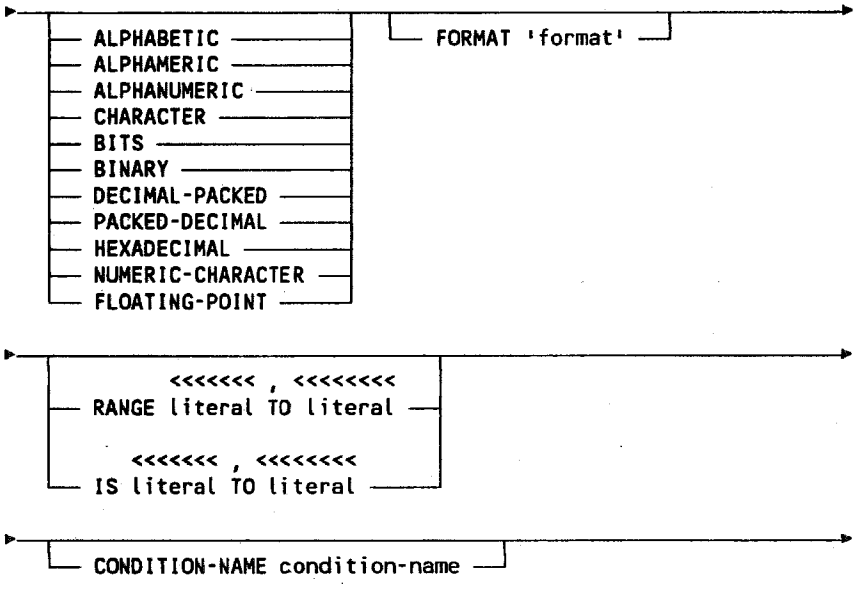
These clauses can also be used in other context.

CONTENTS: Specifying Conditions That Define the Form and Version

This provides a range of stated conditions for defining the form and version of the item more closely. The CONTENTS clause has the form:



content is:



format is a a symbol or a string of not more than 30 characters.

literal may be:

- a delimited character string, or
- a decimal number of less than 18 digits, or
- a floating point number.

If the **form-description** of an ITEM is a Numeric Edited PICTURE clause then the **literal** must be enclosed in quotes.

The following keywords are also valid for use with a literal:

```
HIGH-VALUES, LOW-VALUES, SPACES, QUOTES, ZEROES, ZEROS, ALL.
```

When ALL is specified, a string **'str'** may be used in addition to the above keywords to qualify ALL.

'str' is identical to **literal** except when content is:

```
DECIMAL-PACKED, PACKED-DECIMAL, FLOATING-POINT or NUMERIC-CHARACTER.
```

In these cases, **'str'** is a single numeric character.

Additionally, when **content** is DECIMAL-PACKED, PACKED-DECIMAL, FLOATING-POINT or NUMERIC-CHARACTER, the keywords HIGH-VALUES, LOW-VALUES, SPACES and QUOTES are not valid.

condition-name is a name of not more than 32 characters which must conform to the rules for member names.

The CONDITION-NAME clause is used for the Source Language Generation of COBOL level 88 data description statements.

Any **content** can be specified as conditional, that is, applying only if a stated condition or combination of conditions is satisfied. For a **content** to be conditional, it must be immediately followed by an IF clause. Up to 16 conditional terms may be specified within an IF clause. All contents definitions can have an alternative contents which may also be conditional. Alternative content declarations must be separated by the keyword ELSE. The first conditional term of an alternative content declaration is added to the total number of conditional terms specified in the preceding IF clause.

An item's **form-description** and CONTENTS clause must be compatible. This applies to content-description keywords, FORMAT clause symbols and the literals of both the RANGE and IF clauses.
Examples

```
ITEM
HELD-AS PACKED-DECIMAL 1
CONTENTS IS 0 ELSE RANGE 3 TO 8
;

ITEM
ENTERED-AS ALPHANUMERIC 30
CONTENTS   NUMERIC IF RECORD-TYPE EQ 20
OR RECORD-TYPE EQ 99
ELSE ALPHABETIC IF RECORD-TYPE GT 0
 AND RECORD-TYPE LT 20
ELSE ALPHANUMERIC IF RECORD-TYPE > 20
   AND RECORD-TYPE < 99
;
```

ITEM FORMAT Symbols

The following table lists the symbols used in the FORMAT sub-clause of the CONTENTS clause with a brief explanation of their meaning:

A	Any letter or a space.
N	A numeric digit.
O	A first or last numeric digit which is overpunched if the item is negative.
P	A first or last numeric digit which is overpunched if the item is positive or zero.
S	A space.
X	Any character.
.	A decimal point or a full stop in a date.
/	Forward slash character for use only in a date.
D	Date format; position occupied by the day.
M	Date format; position occupied by the month.
Y	Date format; position occupied by the year.
+	A plus sign if the item is positive, or zero and a minus sign if the item is negative. Must be either the first or last character in a FORMAT which otherwise comprises only N symbols.

- A space if the item is positive, or zero and a minus sign if the item is negative. Must be either the first or last character in a FORMAT which otherwise comprises only N symbols.

FORM-DESCRIPTION	FORMAT SYMBOL:										
	A	N	O	P	S	X	+	-	.	/	DMY
ALPHABETIC	*				*						
ALPHAMERIC	*	*	*	*	*		*	*	*		*
ALPHANUMERIC	*	*	*	*	*		*	*	*		*
BINARY							*	*	*		*
BITS											
CHARACTER	*	*			*	*			*	*	*
DECIMAL-PACKED		*					*	*	*		*
PACKED-DECIMAL		*					*	*	*		*
FLOATING-POINT		*					*	*	*		
HEXADECIMAL											
NUMERIC-CHARACTER		*	*	*			*	*	*		*
PICTURE	*	*	*	*	*	*	*	*	*	*	*

* indicates permissible combinations of symbols and keywords.
Examples

```
ITEM
REPORTED-AS PICTURE '99X99X99'
CONTENTS FORMAT 'DD/MM/YY'
;

ITEM
ENTERED-AS ALPHANUMERIC 7
CONTENTS FORMAT 'NNNNNNX'
;
```

Compatible Form-description and Contents-description Keywords

Form-description	Contents-description	
ALPHABETIC	ALPHABETIC CONDITION-NAME FORMAT	IS RANGE TO
ALPHAMERIC ALPHANUMERIC	ALPHABETIC ALPHAMERIC ALPHANUMERIC CONDITION-NAME FORMAT	IS NUMERIC-CHARACTER RANGE TO
BINARY	BINARY CONDITION-NAME FORMAT	IS RANGE TO
BITS	BITS CONDITION-NAME IS	RANGE TO
CHARACTER	ALPHABETIC ALPHAMERIC ALPHANUMERIC CHARACTER CONDITION-NAME	FORMAT IS NUMERIC-CHARACTER RANGE TO
DECIMAL-PACKED PACKED-DECIMAL	CONDITION-NAME DECIMAL-PACKED FORMAT IS	PACKED-DECIMAL RANGE TO
FLOATING-POINT	CONDITION-NAME FLOATING-POINT FORMAT	IS RANGE TO
HEXADECIMAL	CONDITION-NAME HEXADECIMAL IS	RANGE TO

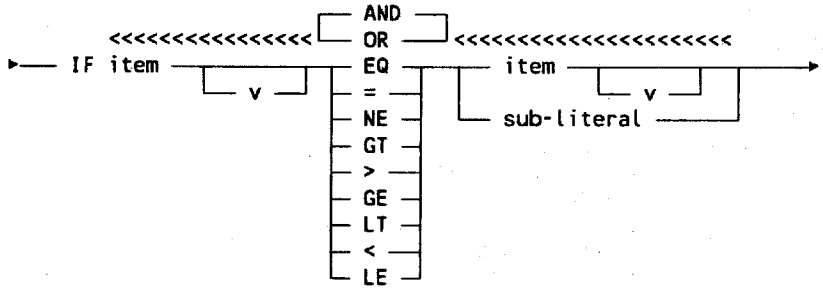
(continued)

(continued)

Form-description	Contents-description	
NUMERIC-CHARACTER	CONDITION-NAME FORMAT IS	NUMERIC-CHARACTER RANGE TO
NAME PICTURE	ALPHABETIC ALPHAMERIC ALPHANUMERIC CONDITION-NAME CHARACTER FORMAT	FLOATING-POINT IS NUMERIC-CHARACTER RANGE TO

The IF Keyword

The IF clause compares the contents of one **item** with the contents of another **item** or a **sub-literal**. Up to sixteen conditional terms may be specified within each IF clause. If there are two or more conditional terms in an IF clause they must be separated by an AND or OR keyword. Any number of IF clauses may appear in an individual member definition. The IF clause has the form:



item is the name of an item

v is an unsigned integer in the range 1 to 15, specifying the version of the associated item. It has a default value of 1.

sub-literal can be one of the following:

- a delimited character string of not more than 256 characters,

- an undelimited signed or unsigned decimal number of not more than 18 digits, optionally with a decimal point,

- an undelimited signed or unsigned floating point number.

If you specify a **sub-literal**, it must be compatible with the item's form-description and the contents-description of its CONTENTS clause.

The operators in IF clauses have the following meanings:

- EQ or = means equal to
- NE means not equal to
- GT or > means greater than
- GE means greater than or equal to
- LT or < means less than
- LE means less than or equal to.

HEADINGS: Specifying a Line of Headings

This optional clause defines a line (or lines) of headings for an item.

For example, in MARK IV Source Language Generation, a PRODUCE command generates file definition forms; the column headings for these forms can be defined in a HEADINGS clause.

The clause has the form:

'string' is a delimited character string; each generates a line of heading.

When processed, the strings are concatenated to form one string. No spaces are inserted in this process, so if any are needed they must be present in the HEADINGS clause.

If a string is preceded by a comma, then it is not concatenated but output to a separate line.

For example:

```
HEADINGS 'DATE OF','ORDER'
```

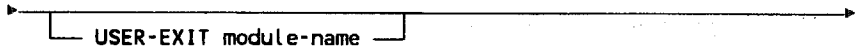
would produce:

```
DATE OF
ORDER
```

USER-EXIT: Specifying the Module in Which the Form and Version is Always Processed

The USER-EXIT clause declares the name of a module in which the form and version of the item is always processed.

The clause has the form:



module-name is the name of a module.

Example

```
ITEM
ENTERED-AS ALPHANUMERIC 7
CONTENTS FORMAT 'NNNNNNX'
USER-EXIT CHKDGT11
;
```

This shows that the ENTERED-AS form of the item is always submitted to a check-digit validation module, CHKDGT11, before further processing.

Examples

```
ITEM
HELD-AS        PACKED 2.2
CONTENTS       RANGE 0 TO 99.99
;

ITEM
REPORTED-AS    PICTURE '+9,999'
CONTENTS       RANGE '-50' TO '+4,999'
;
```

```
ITEM
ENTERED-AS    NUMERIC 6
CONTENTS      FORMAT 'DDMMYY'
REPORTED-AS   PICTURE '99X99X99'
CONTENTS      FORMAT 'DD/MM/YY'
;

ITEM
ENTERED-AS    ALPHANUMERIC 30
CONTENTS      NUMERIC IF RECORD-TYPE EQ 20
OR RECORD-TYPE EQ 99
ELSE ALPHABETIC IF RECORD-TYPE GT 0
  AND RECORD-TYPE LT 20
ELSE ALPHANUMERIC IF RECORD-TYPE > 20
     AND RECORD-TYPE < 99
;
```

OCCURRENCE
To specify the expected number of occurrences of the relationship, enter:

```
OCCURRENCE integer
```

integer is a positive integer within the OCCURRENCE range.

MAXIMUM-OCCURRENCE
To specify the maximum number of occurrences of the relationship, enter:

```
MAXIMUM-OCCURRENCE integer
```

integer is a positive integer.

MINIMUM-OCCURRENCE
To specify the minimum number of occurrences of the relationship, enter:

```
MINIMUM-OCCURRENCE integer
```

integer is a positive integer.

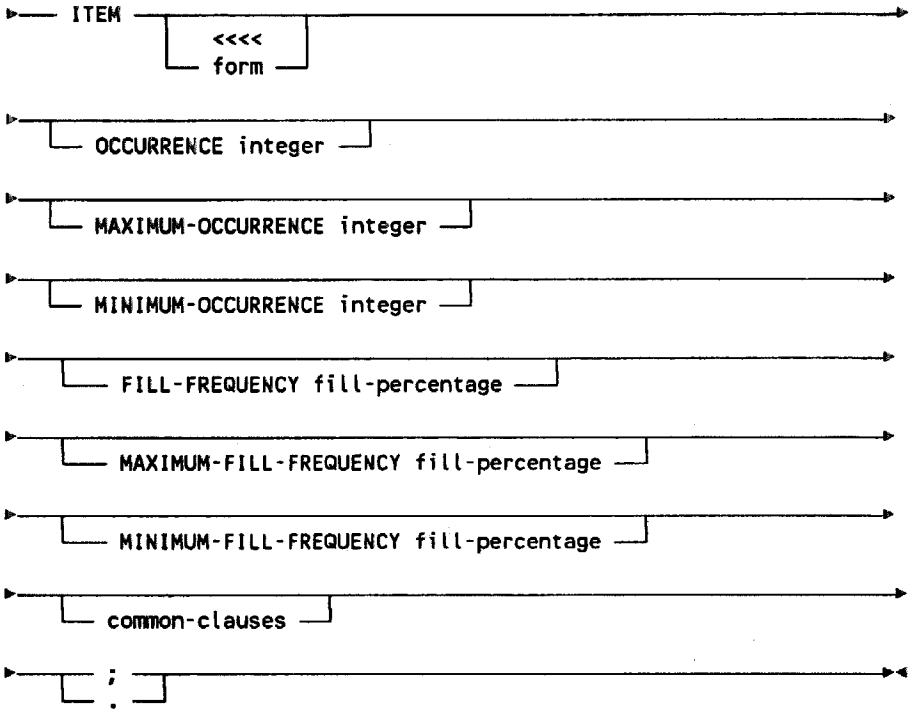
FILL-FREQUENCY
Records the anticipated average percentage of times that this item will contain values. This is relevant to items of information which

are not mandatory, but which may occur frequently.

MAXIMUM-FILL-FREQUENCY
Records the anticipated maximum percentage of times that this item will contain values. This is relevant to items of information which are not mandatory, but which may occur frequently.

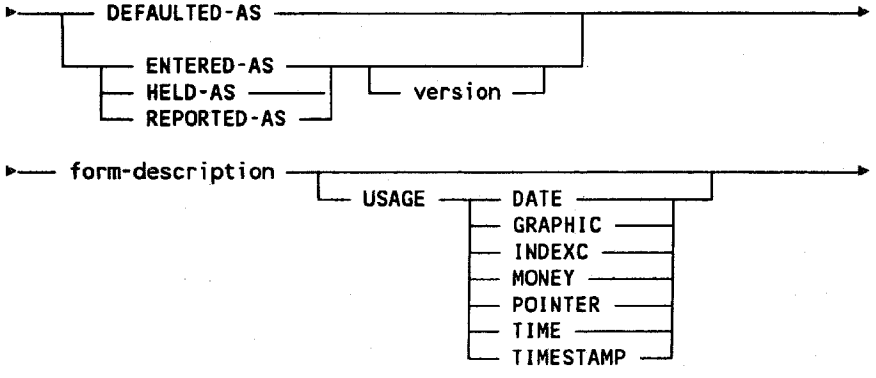
MINIMUM-FILL-FREQUENCY
Records the anticipated minimum percentage of times that this item will contain values. This is relevant to items of information which are not mandatory, but which may occur frequently.

Syntax

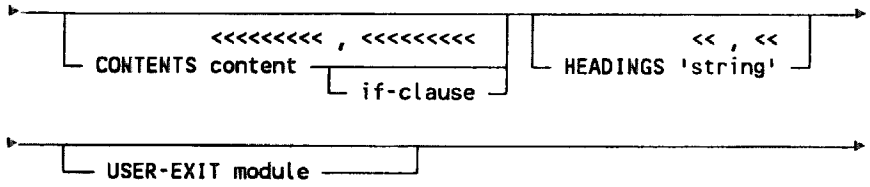


where

form is:



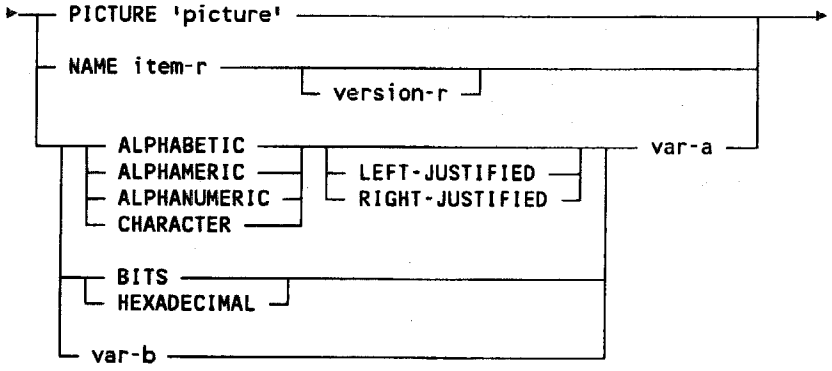
ITEM



where

version is an unsigned integer in the range of 1 to 15.

form-description is:



picture is a symbol or a string of not more than 30 characters. Picture symbols are:

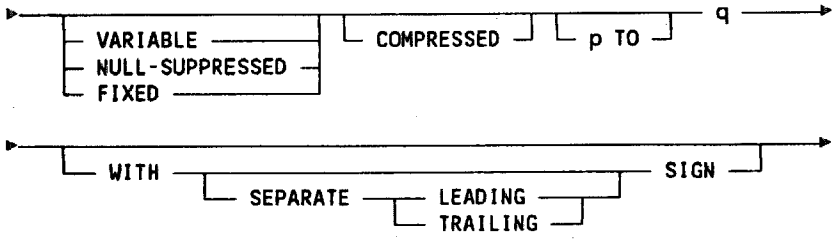
PICTURE TYPE	FORMAT SYMBOL:																						
	A	B	CR	DB	E	I	K	P	R	S	T	V	X	Y	Z	0	9	£	+	–	*	,	.	/
ALPHABETIC	*																							
ALPHANUMERIC	*												*				*							
ALPHANUMERIC EDITED	*	*											*			*	*							
NUMERIC								*				*					*							
NUMERIC EDITED		*	*	*		*		*	*	*	*	*		*	*	*	*	*	*	*	*	*	*	*
NUMERIC FLOATING POINT					*	*	*		*		*	*					*		*	*			*	

* indicates permissible combinations of symbols within picture types.

item-r is the name of an item.

version-r is an unsigned integer in the range 1 to 15.

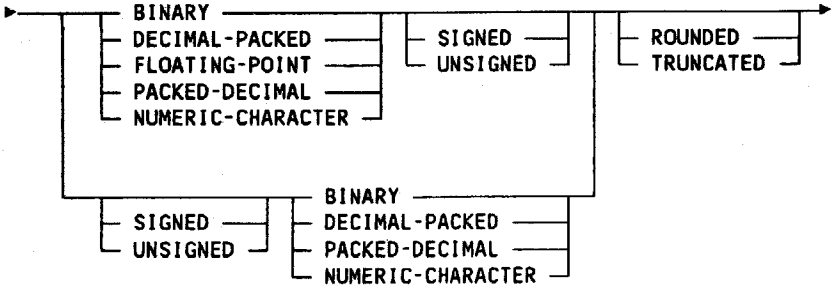
var-a is:



where

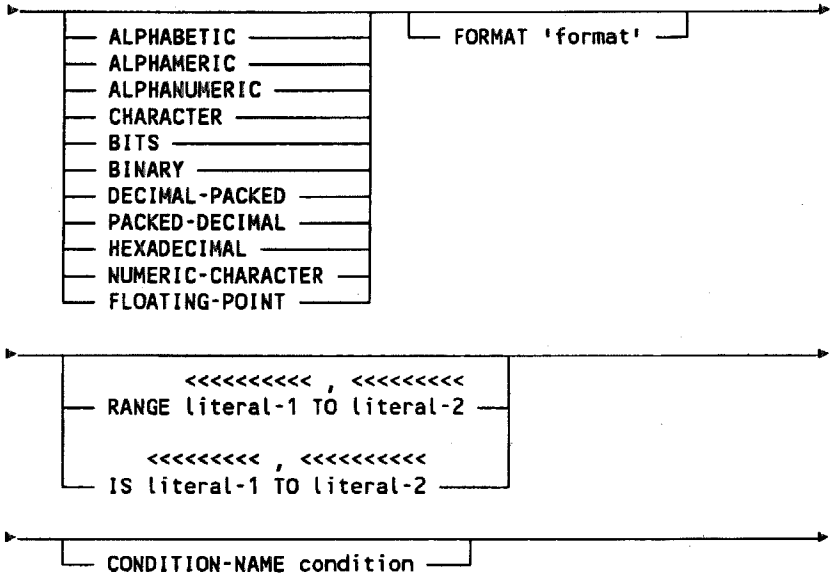
p and **q** are unsigned decimal numbers specifying respectively the minimum and maximum lengths of the item.

var-b is:



ITEM

content is:



'**format**' is a symbol or a string of not more than 30 characters. Format symbols are:

FORM-DESCRIPTION	FORMAT SYMBOL:										
	A	N	O	P	S	X	+	-	.	/	DMY
ALPHABETIC	*				*						
ALPHAMERIC	*	*	*	*	*		*	*	*		*
ALPHANUMERIC	*	*	*	*	*		*	*	*		*
BINARY							*	*	*		*
BITS											
CHARACTER	*	*			*	*			*	*	*
DECIMAL-PACKED		*					*	*	*		*
PACKED-DECIMAL		*					*	*	*		*

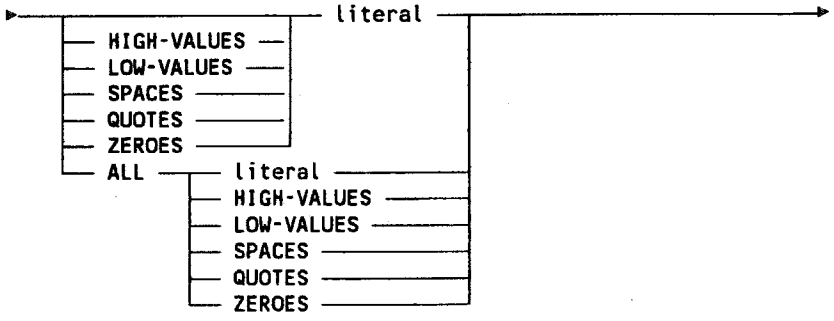
(continued)

(continued)

FORM-DESCRIPTION	FORMAT SYMBOL:										
	A	N	O	P	S	X	+	-	.	/	DMY
FLOATING-POINT		*					*	*	*		
HEXADECIMAL											
NUMERIC-CHARACTER		*	*	*			*	*	*		*
PICTURE	*	*	*	*	*	*	*	*	*	*	*

* indicates permissible combinations of symbols and keywords.

literal-1 and **literal-2** are:



where

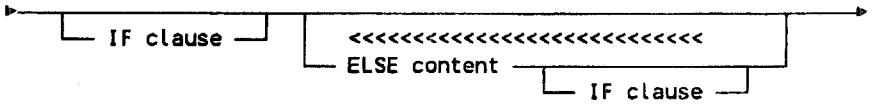
literal is:

- a delimited character string of not more than 256 characters or a numeral in the range 1 to 9
- an undelimited signed or unsigned decimal number of not more than 18 digits, optionally with a decimal point
- an undelimited signed or unsigned floating point number

condition is a name of not more than 32 characters, conforming to the rules for member names.

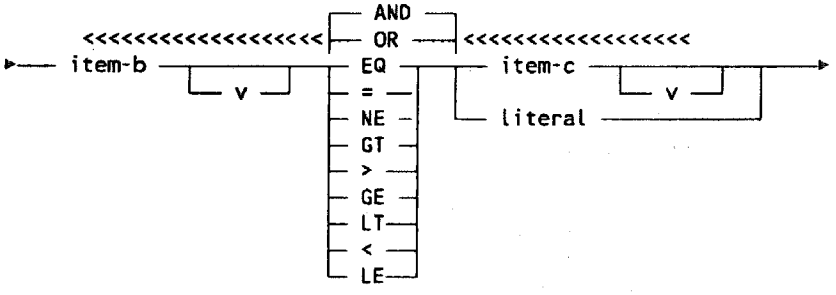
ITEM

if-clause is:



where

clause is:



where

item-b is the name of an ITEM to be compared with item-c or a literal
item-c is the name of the ITEM to be compared with item-b
v is an unsigned integer in the range 1 to 15
literal is the literal being compared with item-b
EQ or **=** means equal to
NE means not equal to
GT or **>** means not greater than
GE means greater than or equal to
LT or **<** means less than
LE means less than or equal to

content is as defined above

'**string**' is a character string of not more than 256 printable characters. Spaces (hexadecimal 40) are regarded as printable characters.

module is the name of a module in which the item is processed.

integer is an integer value of up to 18 digits, optionally preceded by a sign

fill-percentage is an integer in the range 1 to 100

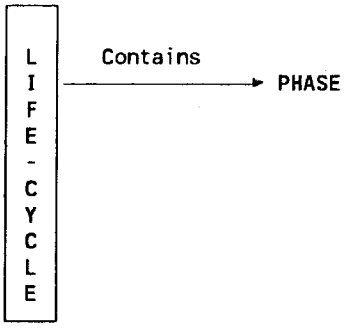
text is a maximum of 32767 delimited character strings, each containing a maximum of 256 characters

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

LIFE-CYCLE

The LIFE-CYCLE constitutes the methodological framework for application development, with reference to method, macro-strategy (life cycle ——▸ PHASES) and micro-strategies (problem solving cycles, ——▸ ACTIVITIES). The latter lead, through the application of particular methods and intermediate results, to a final result set within the framework of a PROJECT.
The more commonly used relationships to and from LIFE-CYCLE are:



CONTAINS
This clause may contain the names of one or more phases.
It must contain the name of every phase which is a part of this life-cycle.
For example:
For a LIFE-CYCLE member whose name is
LC-DEVELOPMENT
you can specify the clause:

```
CONTAINS
  PH-REQUIREMENTS
, PH-ANALYSIS
, PH-DESIGN
, PH-CODE
, PH-TEST
, PH-INSTALL
```

OPTION
Short description of project management components, as shown on the user interface.

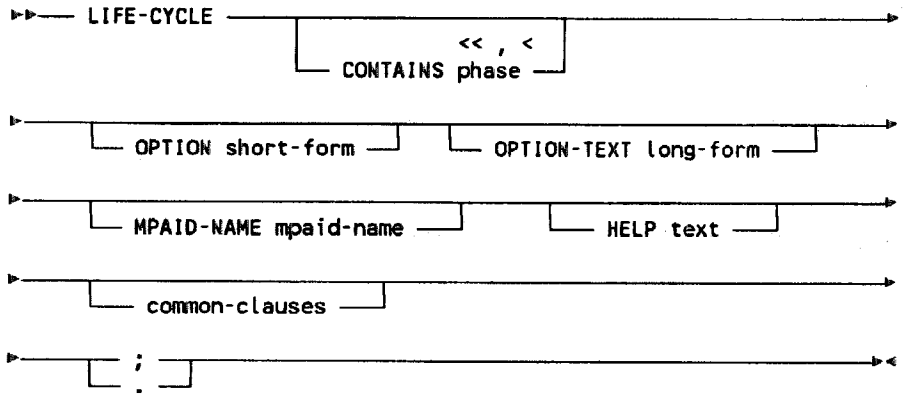
OPTION-TEXT
Long description of project management components, as shown on the user interface.

MPAID-NAME
MPAID-NAME for the generation of the life cycle.

HELP
Description of the member with no limit on length.

LIFE-CYCLE

Syntax



where

phase is the name of a PHASE member

short-form is a delimited string of up to 32 characters

long-form is a delimited string of up to 50 characters

mpaid-name is a name of up to 10 characters

text is a maximum of 32767 delimited character strings, each containing a maximum of 256 characters

common-clauses are any of the clauses common to all member types.

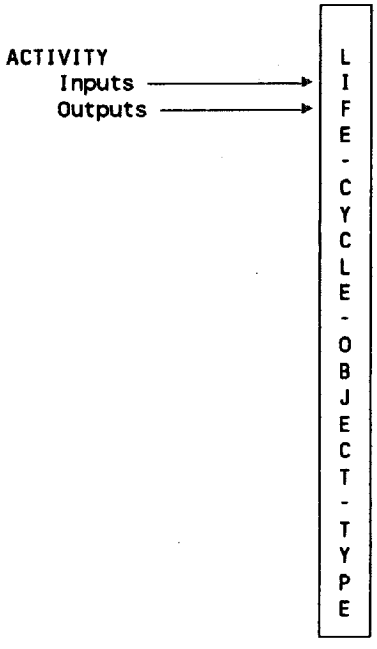
Refer to Appendix 2 for details of the common clauses.

LC-OBJECT-TYPE

A LIFE-CYCLE-OBJECT-TYPE is a formal description of a result. It is possible that only part of the result is produced by one ACTIVITY and another part by another activity.

They are classified according to where they are stored.
The tool corresponding to the LIFE-CYCLE-OBJECT-TYPE can be called to create or process the result.

The more commonly used relationships to and from LIFE-CYCLE-OBJECT-TYPE are:



OPTION
Short description of project management components, as shown on the user interface.

OPTION-TEXT
Long description of project management components, as shown on the user interface.

LIFE-CYCLE-OBJECT-TYPE

MPAID-NAME
MPAID-NAME for the generation of the life cycle.

DATA-STORE
Results must be stored in one of the following areas:
Repository, Extern, Screen, Workbench

TYPE
Indicates whether or not the result must be present/created.

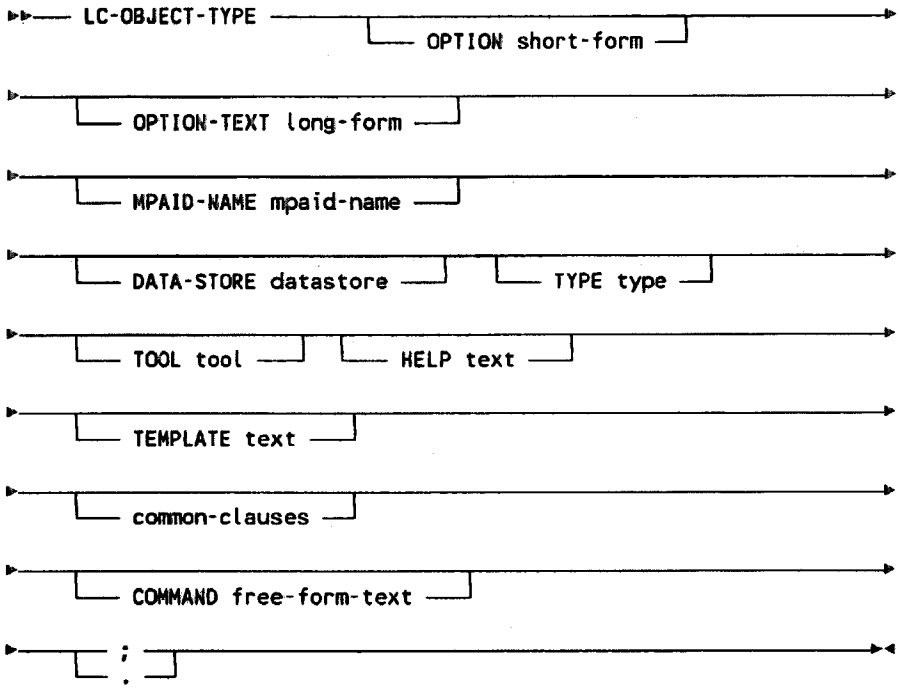
TOOL
Name of the tool used to create a result of this type. The name must correspond to the short name for the tool given in the selection panel.

HELP
Description of the member with no limit on length.

TEMPLATE
Formatted document layout

COMMAND
A series of commands can be given, which are executed should the 's' selection command be used.

Syntax



where

short-form is a delimited string of up to 39 characters

long-form is a delimited string of up to 50 characters

mpaid-name is a name of up to 10 characters

datastore is one of the following:

```
DICTIONARY
EXTERNAL
SCREEN
WORKBENCH
```

type is:

O for optional

M for mandatory

LIFE-CYCLE-OBJECT-TYPE

tool is a delimited string of up to 15 characters

text is a maximum of 32767 delimited character strings, each containing a maximum of 256 characters

common-clauses are any of the clauses common to all member types.

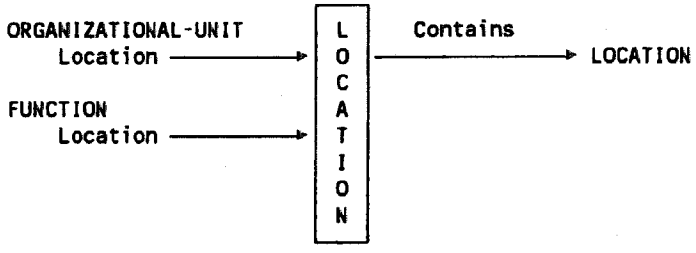
Refer to Appendix 2 for details of the common clauses.

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 248 characters.

LOCATION

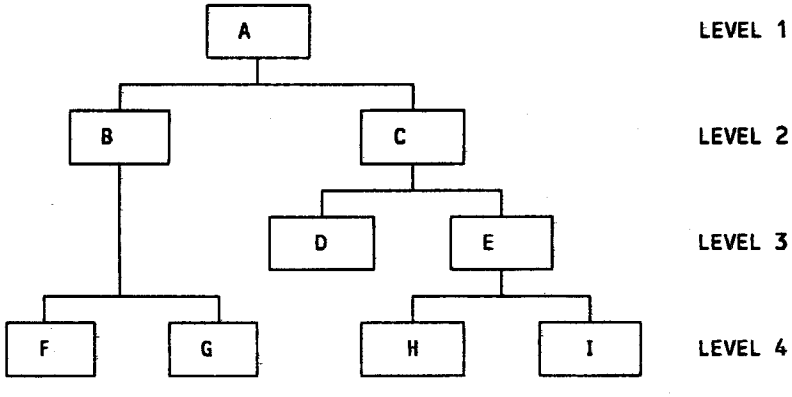
A LOCATION is the geographical place or area where an organizational unit exists.

The more commonly used relationships to and from LOCATION are:



CONTAINS
Contains the name of a location which is within this location.

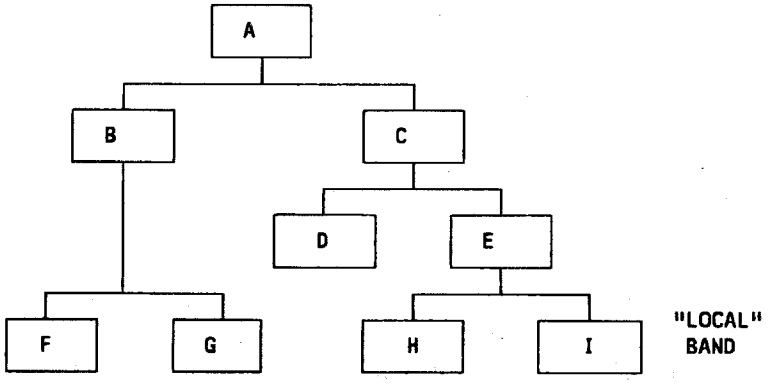
LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy.
The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

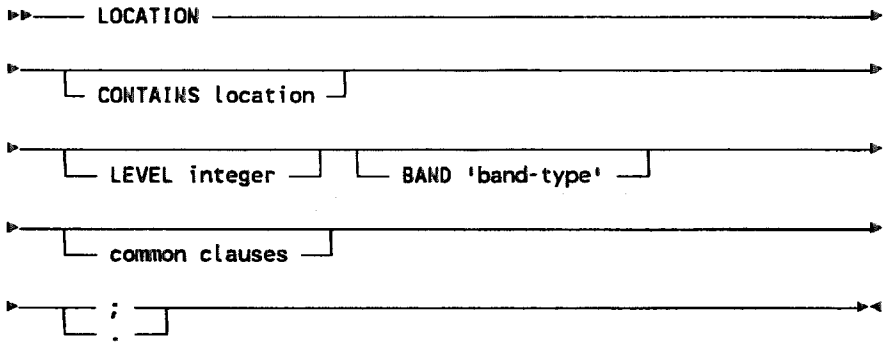
BAND

The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.



The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

Syntax



where

location is the name of a LOCATION member

integer is an integer value of up to 18 digits, optionally preceded by a sign

'band-type' is a text string of up to 78 characters, including delimiters.

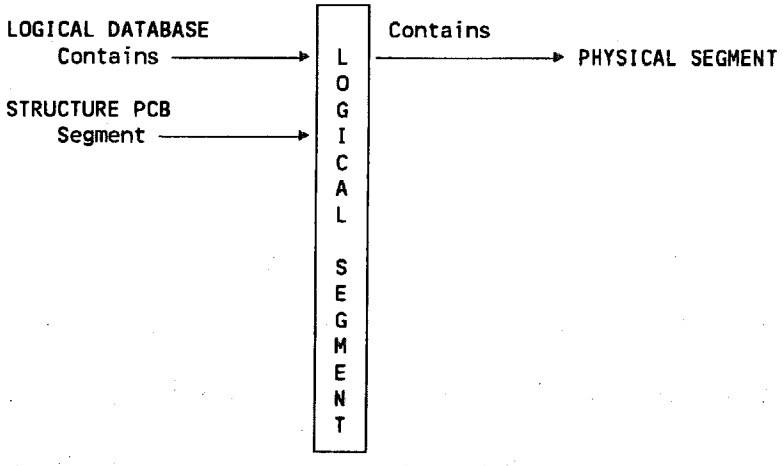
Note: the delimiters shown in the above syntax are required when defining a LOCATION member via the command interface.

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

LOGICAL-SEGMENT

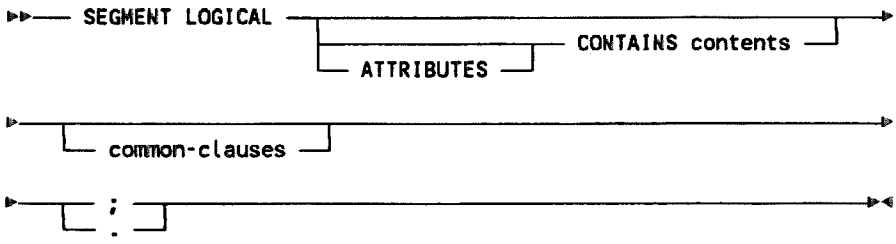
A LOGICAL SEGMENT may contain a physical segment or the concatenation of two physical segments.
The more commonly used relationships to and from LOGICAL SEGMENT are:



For further details refer to the "IMS (DL/1) Interface" documentation.

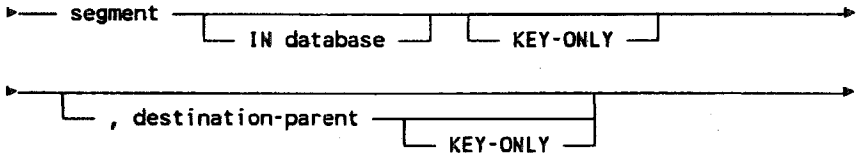
ATTRIBUTES CONTAINS
The logical segment may contain a physical segment or a concatenation of logical child and destination parent segments.

Syntax



where

contents are:



where:

segment is the name of a PHYSICAL SEGMENT

database is the name of a HISAM, HDAM or HIDAM database

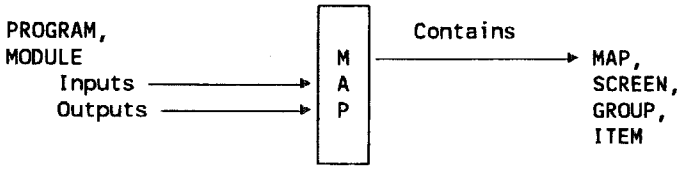
destination-parent is a PHYSICAL DESTINATION-PARENT-SEGMENT.

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

MAP

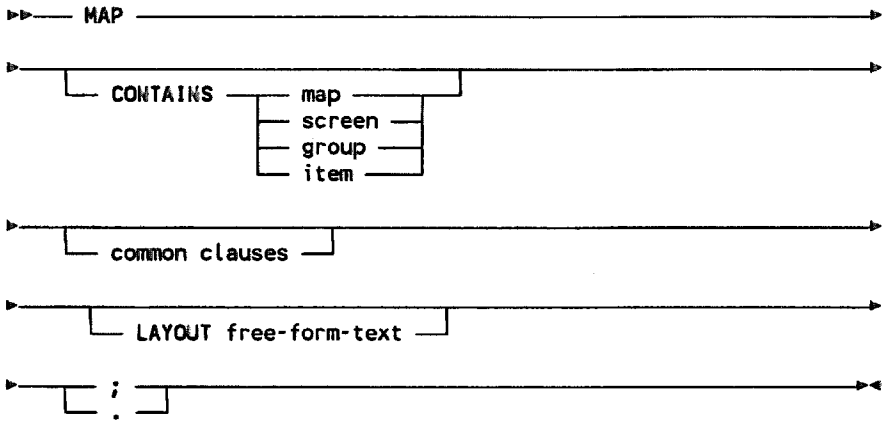
A MAP is a set of data elements and literals comprising a logical screen image which may represent more than one full physical screen of data or less. It also includes layout field attributes and other relevant information.
The more commonly used relationships to and from MAP are:



CONTAINS
Contains the name of any MAP, SCREEN, GROUP or ITEM members which the map uses to create a logical screen image.

LAYOUT
Contains free-form text, and must therefore be the last clause in the member. It is used to establish the exact layout required for the map, such as the positions of protected and unprotected fields.

Syntax



where

map is the name of a MAP member

screen is the name of a SCREEN member

group is the name of a GROUP member

item is the name of an ITEM member

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 248 characters.

MARKIV

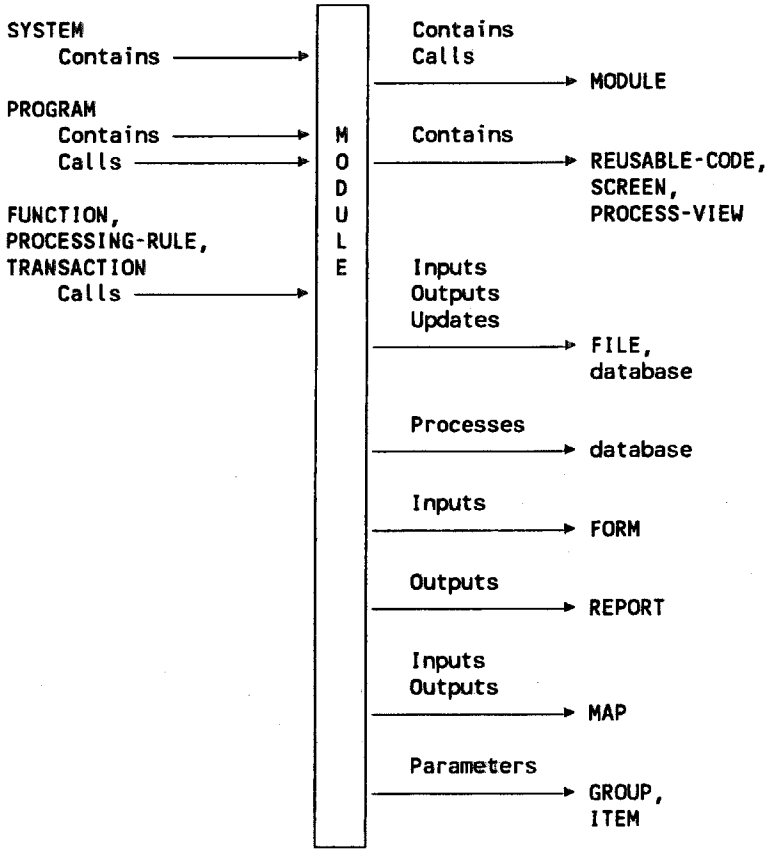
For details of the MARK IV interface member types:

- MARKIV-FILE
- MARKIV-SEGMENT

please refer to the MARK IV Interface manual (DMR-MKIV).

MODULE

A MODULE is a program unit that is discrete and identifiable with respect to compiling, combining with other units and loading but cannot be executed on its own.
The more commonly used relationships to and from MODULE are:



CONTAINS

Contains the names of any PROCESS-VIEW, MODULE, REUSABLE-CODE, SCREEN, FILE, GROUP or ITEM members used by this module.

MODULE

CALLS
This clause may contain the name of one or more MODULE members. It may also contain one AT subclause, naming the point at which the module is called, and a PASSING clause naming the data members passed to it. Only one AT subclause can be specified in a CALLS clause. To define multiple called modules, each with its own AT subclause, multiple CALLS clauses must be specified.

INPUTS
Contains the names of any FILE, GROUP or ITEM members which input information to this member for processing or reference. It should also list the names of members updated by generation of a new version, rather than replacement of an existing version.

OUTPUTS
Lists the names of members to which this module writes information used for processing or reference. It should also list the names of members which are updated by generating a new version, rather than replacing an existing version.

UPDATES
The only FILE, GROUP or ITEM members named here are those which this module updates by replacing the original version. Members which are updated by generation of a new version should be declared in the INPUTS and OUTPUTS clauses of the module.

PARAMETERS
Contains the names of ITEM, GROUP or FILE members which define parameters passed to this member. It may also include an ENTRY-POINT subclause for each data-name, containing the label of an entry point in this member.

PROCESSES
Contains the name of each database processed through this process-view. Each database whose processing is described in the DESCRIPTION clause of this member must be named in this clause. A separate PROCESSES attribute may specify each database processing requirement in free format.

Note: for IMS processing the PROCESSES IMS CONTAINS attribute should be used to specify the constituent database PCB member names.

PROCESSES IMS CONTAINS
Contains the name of each database PCB referenced through this process-view.

SEGMENT-SEARCH-ARGUMENTS
May contain a list of Segment Search Arguments (SSAa) for sensitive segments contained by the Program Communication Blocks (PCBs) which are referenced by this program.

Multiple USED-IN clauses may be specified where there are many SSAs for a segment.

SSAS
Will only appear for existing PROGRAM members if the keyword SSAS has been specified as an alternative to SEGMENT-SEARCH-ARGUMENTS.

AUTHOR
Records the name of the author of this member. It is needed if a COBOL source program is generated from this definition. Otherwise the clause is used for general documentation.

LANGUAGE
The name of the computer language in which the member is programmed.

INSTALLATION
Contains the name of the installation where this member is used. It is needed if a COBOL source program is to be generated from this definition. Otherwise the clause is used for general documentation.

DATE-WRITTEN
Contains a date, in your organization's standard format, stating when this member was set up. It is needed if a COBOL source program is generated from the member. Otherwise it is used for general documentation.

SOURCE-COMPUTER
Contains the name of the computer that is the source of data used in this member. This clause is needed if a COBOL source program is to be generated from the definition. Otherwise it is used for general documentation.

OBJECT-COMPUTER
Contains the type of the computer on which the object module created this program or module will run.
It is needed if a COBOL object program is to beGenerated from this definition. Otherwise it is used for documentation.

SPECIAL-NAMES
Forces the processing code to replace a specific character with another character. This clause is needed if a COBOL object program is to be generated from the member which contains this clause. Otherwise it is used for general documentation.

I-O-CONTROL
Stores any special I-O instructions for the member being defined. It is needed if a COBOL object program is to be generated from this definition. Otherwise the clause is used for general documentation.

ASSIGNMENT
Contains the name of the device assigned to this member. It is needed if a COBOL object program is generated from this definition. Otherwise the clause is used for general documentation.

EDIT-INPUT
Contains instructions about editing input to the member before processing. It is needed if a COBOL source program is to be generated from this member. Otherwise it is used for documentation.

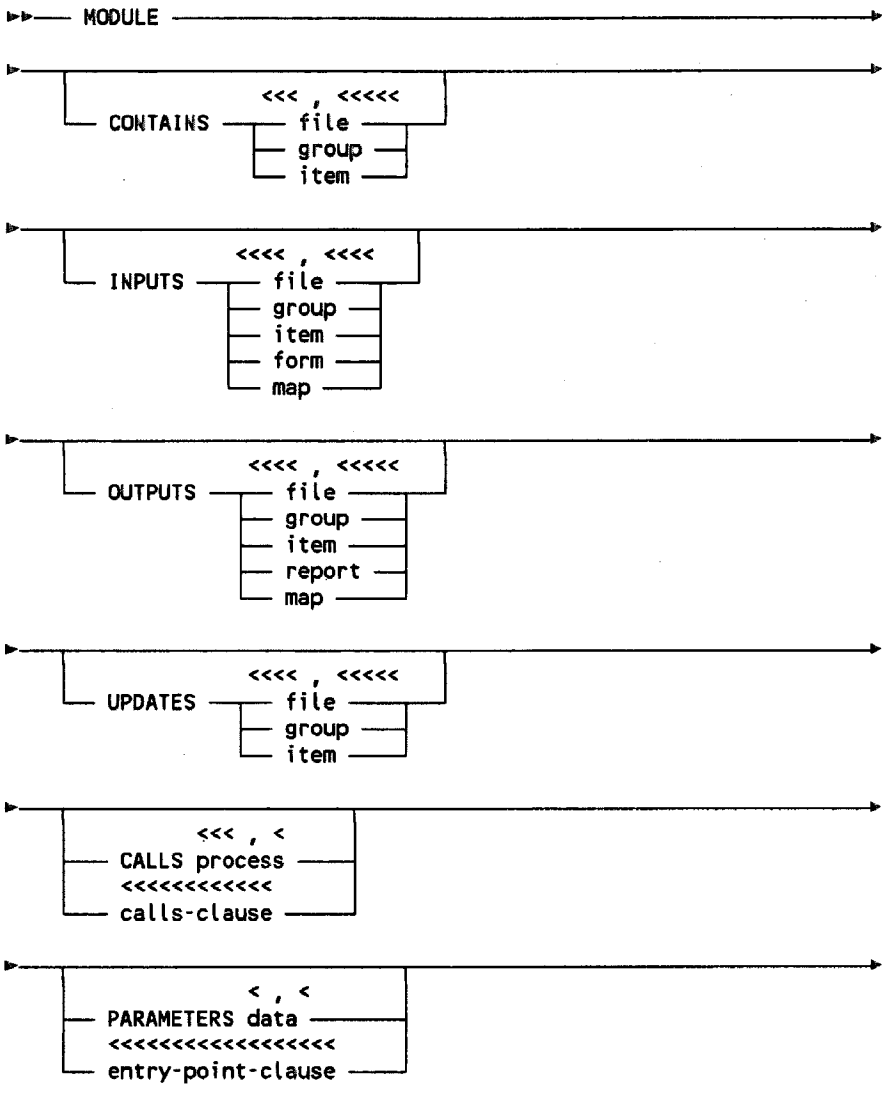
EDIT-OUTPUT
Contains instructions about any editing of output to be done before this member begins processing. It is needed if a COBOL source program is to be generated from this definition.
Otherwise the clause is used for general documentation.

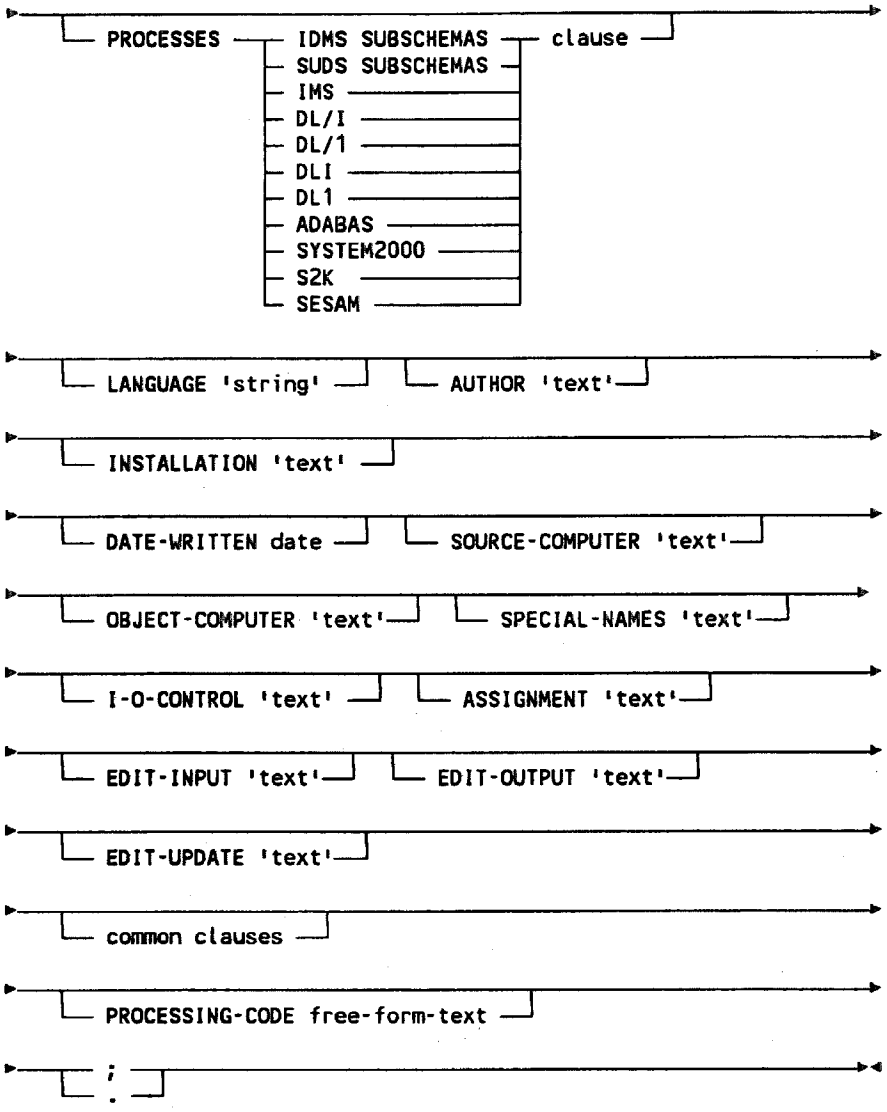
EDIT-UPDATE
Contains instructions about anything which needs editing after being updated by this member. It is needed if a COBOL source program is to be generated from this definition. Otherwise the clause is used for general documentation.

PROCESSING-CODE
Contains processing code as free-form text, and must be the last clause in the member. It is needed if a COBOL source program is to be generated from this definition. Otherwise it is used for general documentation.

MODULE

Syntax





where

file is the name of a FILE member

group is the name of a GROUP member

item is the name of an ITEM member

MODULE

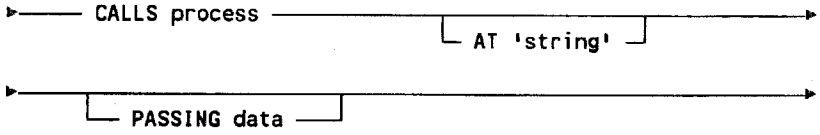
form is the name of a FORM member

map is the name of a MAP member

report is the name of a REPORT member

process identifies a process member at the same or a lower level in the member type hierarchy

calls-clause is:



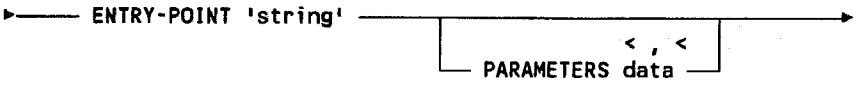
where

process is as defined above

'string' is a character string of not more than 256 characters

data identifies a data member

entry-point-clause is:



where **'string'** and **data** are as defined above

clause is defined in the User Guide specific to your environment.

'string' is as defined above

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters

Note: the delimiters shown in the above syntax are required when defining a MODULE member via the command interface.

date is a date, in the format defined by your installation

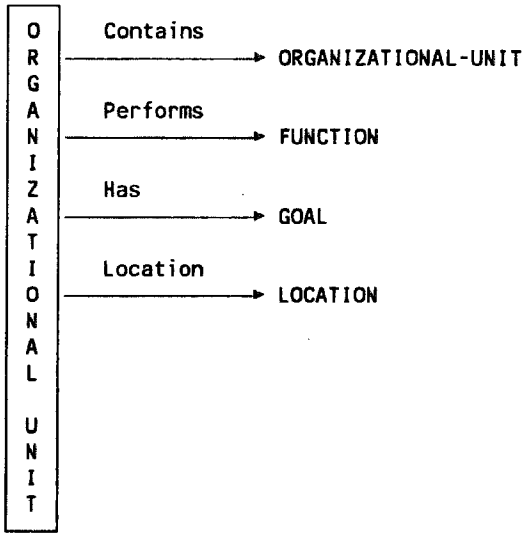
common clauses are any of the clauses common to all member types.

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 248 characters.

Refer to Appendix 2 for details of the common clauses.

ORGANIZATIONAL-UNIT

An ORGANIZATIONAL-UNIT is the organization or part thereof which can be considered as a self-contained unit for operations, duties, responsibilities and authorities.
The more commonly used relationships to and from ORGANIZATIONAL-UNIT are:



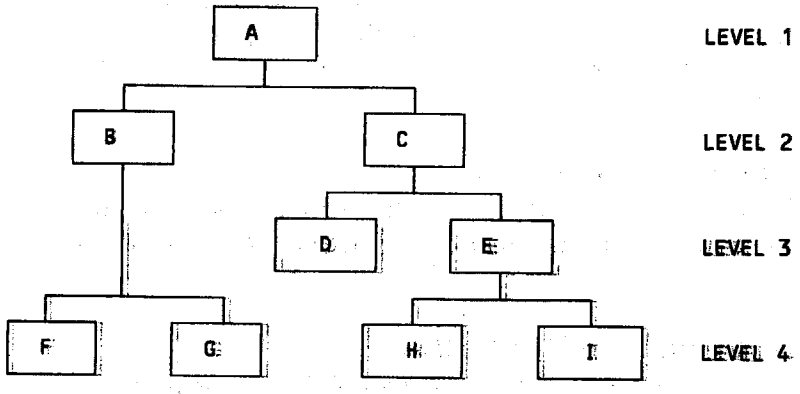
CONTAINS
Contains the names of the organizational units which make up this member.

PERFORMS
Lists the functions performed by this organizational-unit. An optional INVOLVEMENT subclause indicates the degree of involvement in performing the function.

HAS
This clause contains the names of the goals assigned to this member.

LOCATION
Defines the location within the enterprise of the organizational-unit being defined, by naming the appropriate LOCATION members.

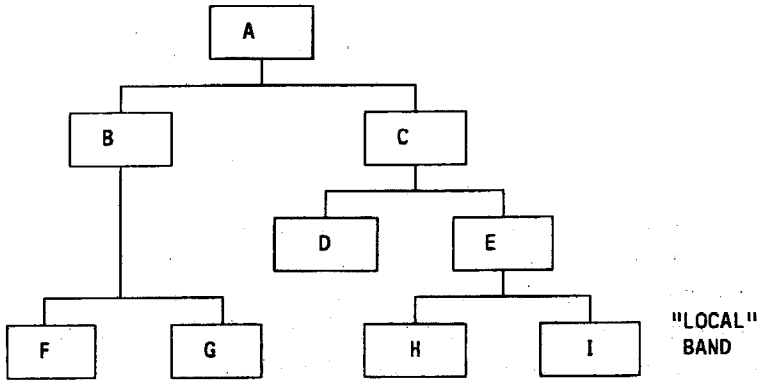
LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy.
The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

BAND
The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.

ORGANIZATIONAL-UNIT

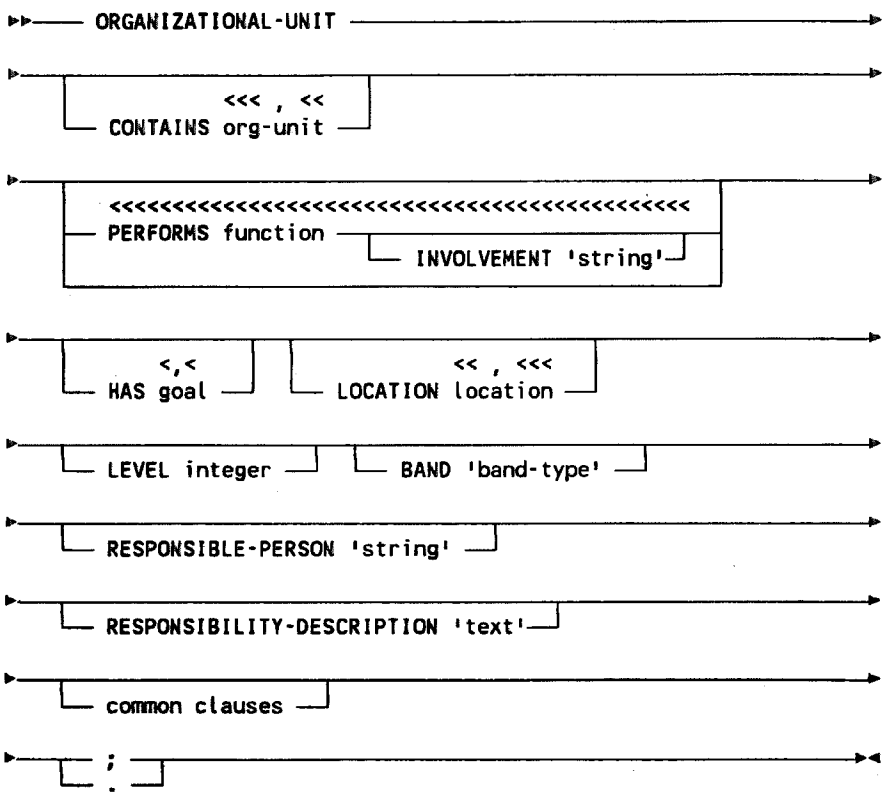


The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

RESPONSIBLE-PERSON
Identifies the person responsible for the organizational unit. It is recommended that the person's job title is used rather than his or her name.

RESPONSIBILITY-DESCRIPTION
Describes the duties and responsibilities of the person named in the RESPONSIBLE-PERSON clause of this ORGANIZATIONAL-UNIT member.

Syntax



where

org-unit is the name of an ORGANIZATIONAL-UNIT member

function is the name of a FUNCTION member

goal is the name of a GOAL member

location is the name of a LOCATION member

integer is an integer value of up to 18 digits, optionally preceded by a sign

'band-type' is a text string of up to 78 characters, including delimiters.

'string' is a character string of not more than 256 characters

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters.

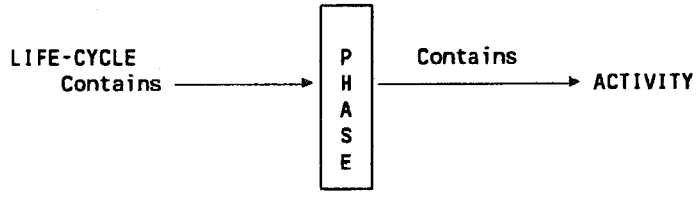
Note: the delimiters shown in the above syntax are required when defining an ORGANIZATIONAL-UNIT member via the command interface.

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

PHASE

PHASEs give the life cycle a structure (macro strategy) and are themselves structured through ACTIVITIES (micro strategies), the latter leading to a phase end which is planned and controlled by a PROJECT. The more commonly used relationships to and from PHASE are:



CONTAINS
This clause may contain the names of one or more activities
It must contain the name of every activity which is a part of this phase.
For example:
For a PHASE member whose name is

```
PH-REQUIREMENTS
```

you can specify the clause:

```
CONTAINS
    AC-INTERVIEW-USERS
  , AC-CONSOLIDATE-REQUIREMENTS
```

OPTION
Short description of project management components, as shown on the user interface.

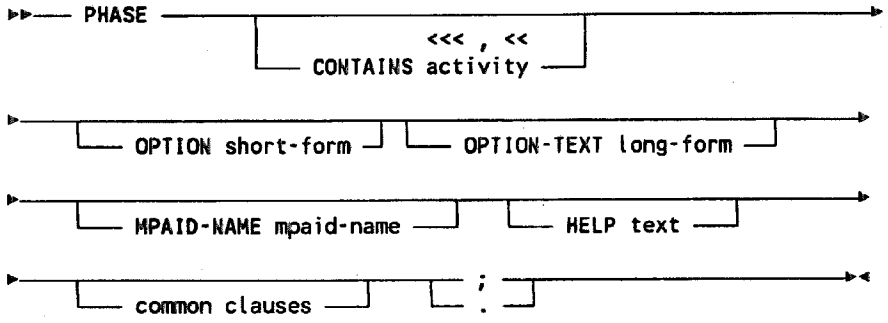
OPTION-TEXT
Long description of project management components, as shown on the user interface.

MPAID-NAME
MPAID-NAME for the generation of the life cycle.

PHASE

HELP
Description of the member with no limit on length.

Syntax



where

activity is the name of an ACTIVITY member

short-form is a delimited string of up to 32 characters

long-form is a delimited string of up to 50 characters

mpaid-name is a name of up to 10 characters

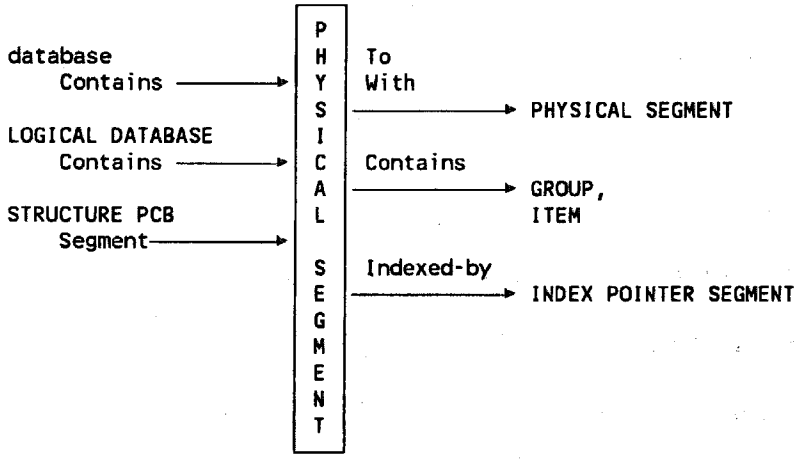
text is a maximum of 32767 delimited character strings, each containing a maximum of 256 characters

common-clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

PHYSICAL-SEGMENT

A PHYSICAL SEGMENT defines the structure of a database hierarchy. The more commonly used relationships to and from PHYSICAL SEGMENT are:



For further details refer to the "IMS (DL/1) Interface" documentation.

RELATED-AS
Specify what type of segment this is with regard to logical relationships and secondary indexes. INSERT, DELETE, and REPLACE rules may be specified as PHYSICAL, LOGICAL, or VIRTUAL. POINTERS may be specified as SYMBOLIC or DIRECT-ADDRESS, and for a REAL-PAIRED-CHILD-SEGMENT the logical relationship pointer options may be specified as FORWARD-LOGICAL-TWIN or BACKWARD-LOGICAL-TWIN and SINGLE-LOGICAL-CHILD or DOUBLE-LOGICAL-CHILD. Where appropriate a CONCATENATED-KEY-NAME may be specified with an optional RENAMES clause for local names. The TARGET and SOURCE segment keywords are required if this segment is the target of or contains source for a secondary index pointer segment.
If the segment is referred to by more then one database then this clause should be deleted.

ATTRIBUTES CONTAINS
Here the constituent GROUP/ITEMs of a segment may be specified within the CONTAINS clause.

FREQUENCY
The number of segments held per parent, or an integer for the number of root segments (for child segments there may be two decimal points).

SEQUENCE-KEY
Defines the sequence key for the segment.

INSERT-POSITION
Defines where the segment is inserted amongst its siblings.

POINTERS
i.e. SINGLE-TWIN/DOUBLE-TWIN/FORWARD-HIERARCHICAL etc.
FIRST-CHILD/LAST-CHILD
COUNTER

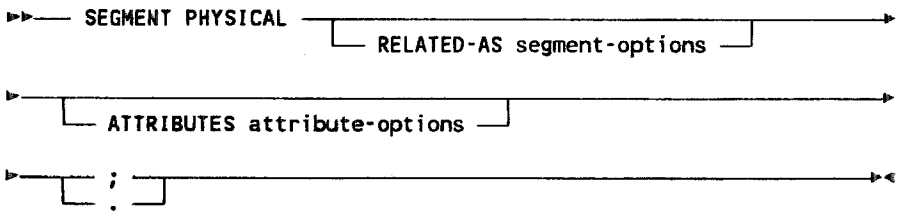
EDIT-COMPRESSION-EXIT
Defines the exit and parameters ALL and OPEN/CLOSE if required.

CHANGED-DATA-CAPTURE-FACILITY
Defines the DCDCF exit list with any options, or specifies NOT-USED/UNUSED to bypass DCDCF.

GENERATES
Defines generated fields if required.

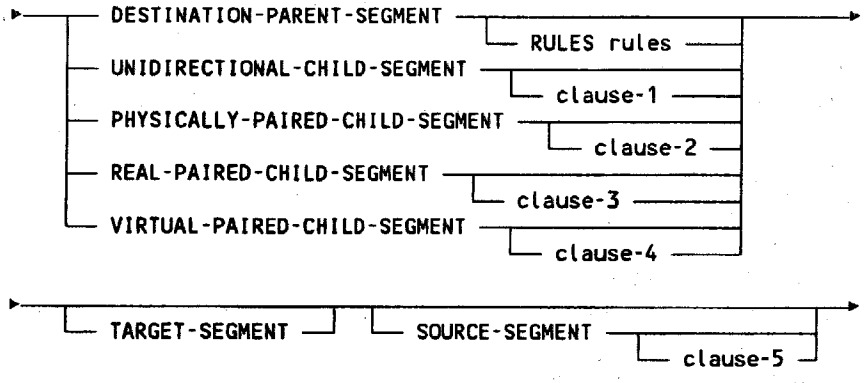
PHYSICAL-SEGMENT

Syntax



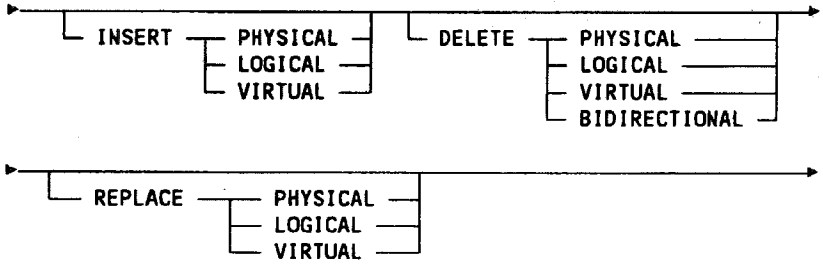
where

segment-options are:

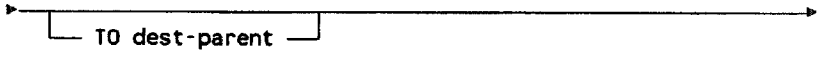


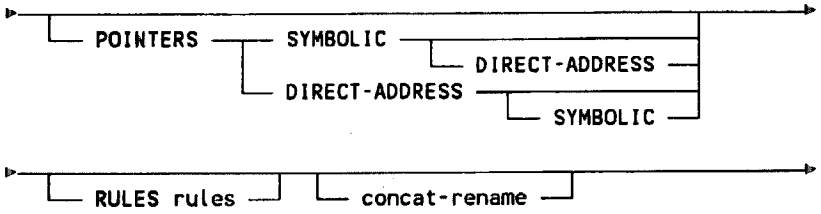
where

rules are:



clause-1 is:

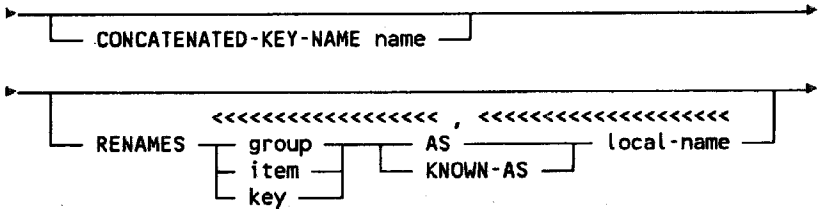




where

dest-parent is the name of a PHYSICAL DESTINATION-PARENT-SEGMENT member

concat-rename is



where

name is the name of a CONCATENATED-KEY member

group is the name of a GROUP member

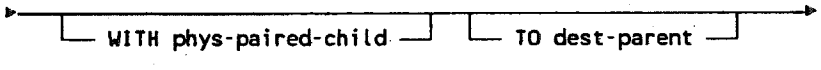
item is the name of an ITEM member

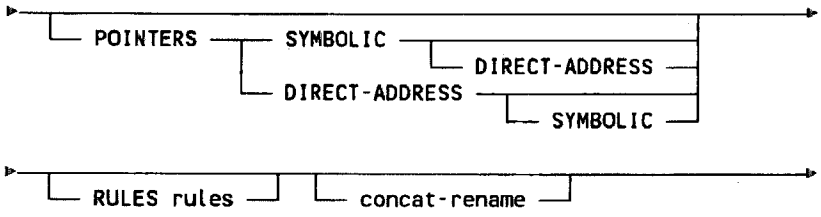
key is a 1 to 8 character unique alphanumeric name

local-name is a name, conforming to the rules for member names as stated in the CONTROLMANAGER User's Guide (CMR-UG)

rules are as defined above.

clause-2 is:



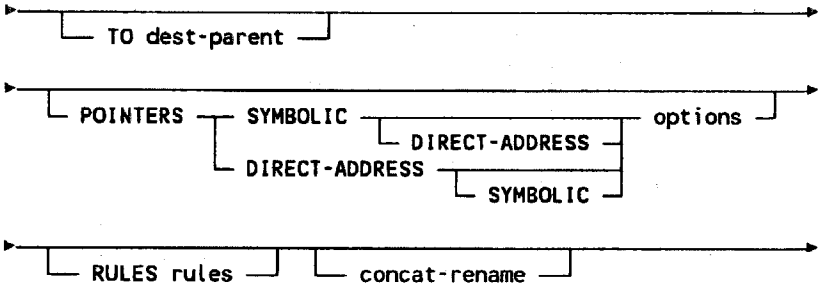


where

phys-paired-child is the name of a PHYSICALLY-PAIRED-CHILD-SEGMENT member

dest-parent, rules, concat-rename are as defined above.

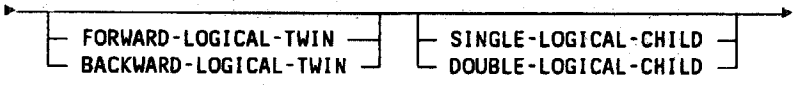
clause-3 is:



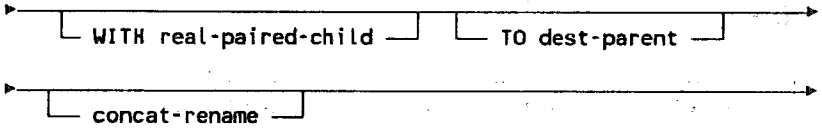
where

dest-parent, concat-rename, phys-paired-child and **rules** are as defined above

options are:



clause-4 is:

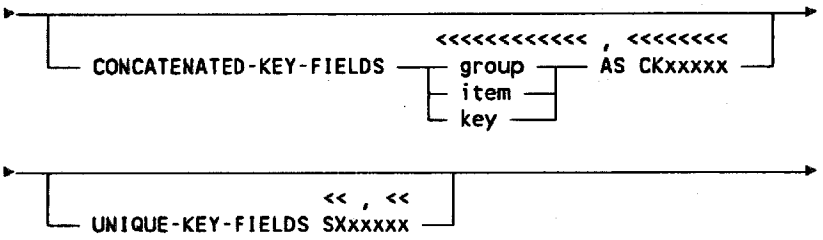


where

real-paired-child is the name of a PHYSICAL REAL-PAIRED-CHILD-SEGMENT member

dest-parent and **concat-rename** are as defined above

clause-5 is:



where

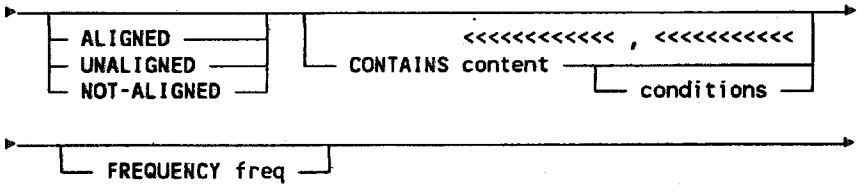
group, item and **key** are as defined above

CKxxxxx is a 3 to 7 character name to be applied to a system related field within a GENERATES, SUBSEQUENCE-FIELDS or DUPLICATE-DATA-FIELDS clause. The name must be unique and start with CK.

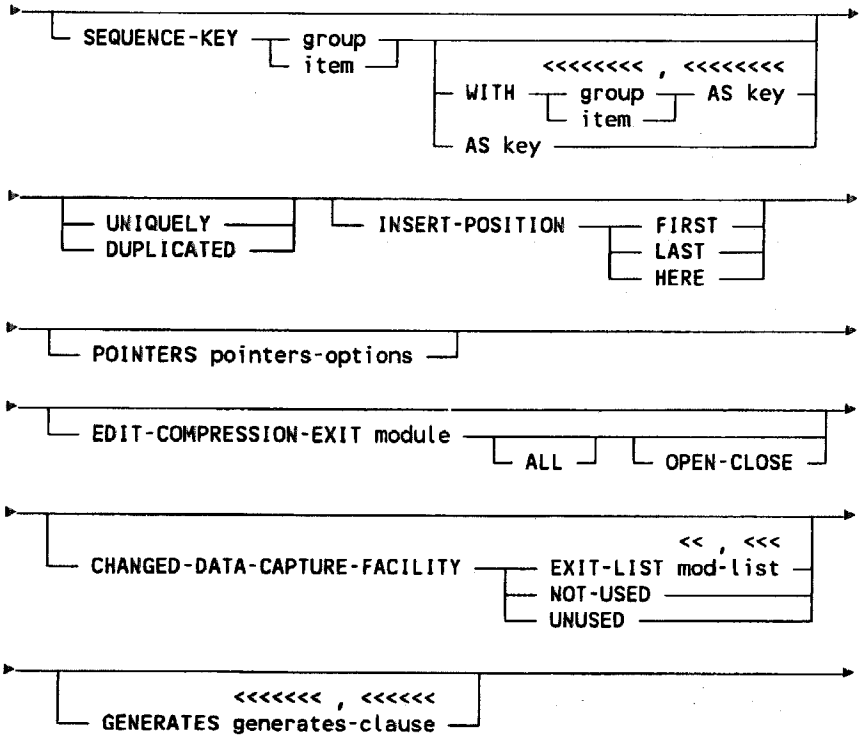
SXxxxxx is a 3 to 7 character name to be applied to a system related field within a GENERATES or SUBSEQUENCE-FIELDS clause. The name must be unique and start with SX.

key-name is a 1 to 8 character unique alphanumeric name

attribute-options are:

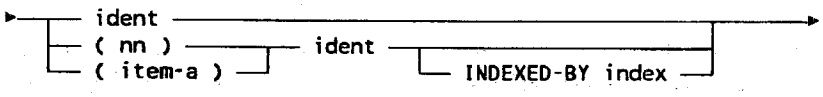


PHYSICAL-SEGMENT

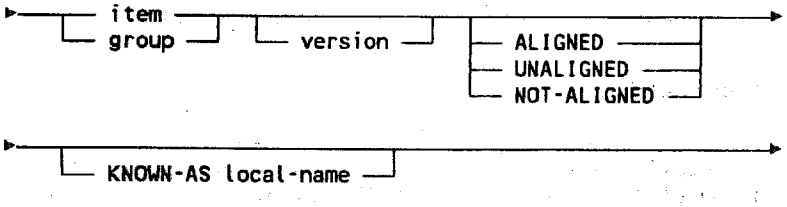


where

content is:



where **ident** is:



where:

item, group are as defined above

version is an unsigned integer in the range 1 to 15

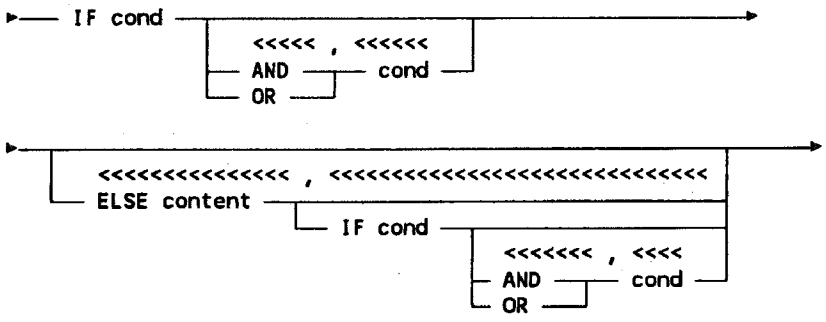
local-name is as defined above

nn is an unsigned integer of from 1 to 18 digits, being the number of times **item** or **group** occurs in the array

item-a is the name of an ITEM member

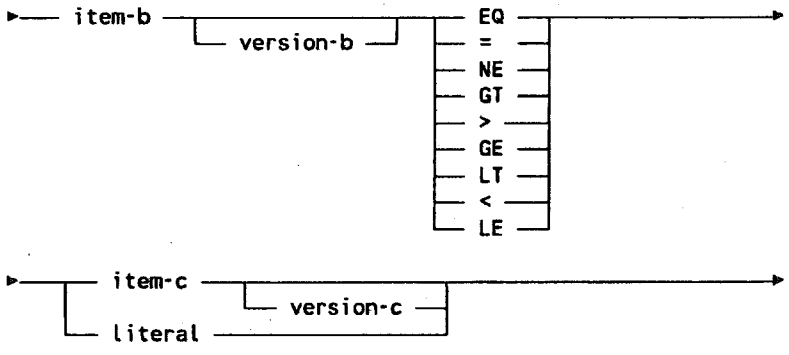
index is a name, conforming to the rules for member names

conditions are:



where

cond is:



where:

literal is a literal comparand

item-b is the name of the ITEM whose contents are to be compared with the comparand

version-b is an unsigned integer in the range 1 to 15

item-c is the name of the ITEM whose contents are the comparand

version-c is an unsigned integer in the range 1 to 15

freq is an unsigned number in the range 0.01 to 16777215.00, or (for root segments) an integer in the range 1 to 16777215

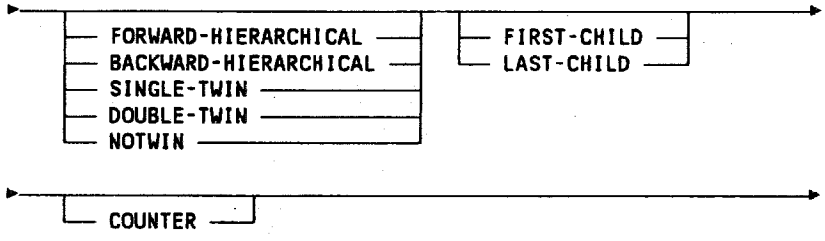
content is as defined above

name is the name of a CONCATENATED-KEY member

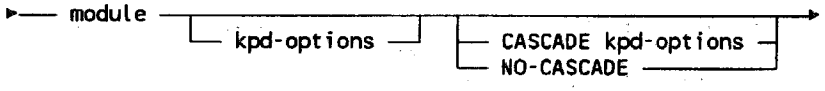
module is the name of a MODULE member.

freq, group, item, key and **module** are as defined above

pointers-options are:



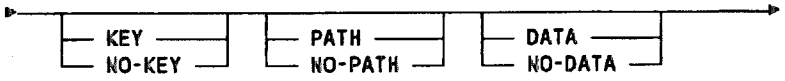
mod-list is:



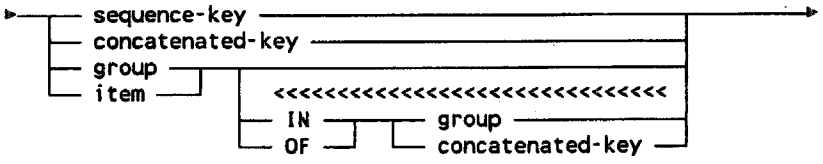
where:

module is the name of a MODULE or PROGRAM member. Dummies are created as modules.

kpd-options are:



generates-clause is:



where:

concatenated-key specifies the name to be given to the destination parent's concatenated key

group, item are as defined above

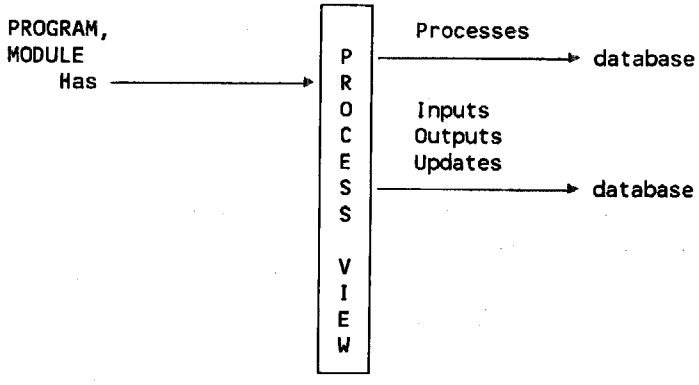
sequence-key is a 1 to 8 character unique alphanumeric name.

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

PROCESS-VIEW

A PROCESS-VIEW is a description of the processing of one or more databases by one or more programs or modules.
In an IMS environment the name PSB may be used instead of process view but, in this case, the user must ensure that there is only one PSB for each program.
The more commonly used relationships to and from PROCESS-VIEW are:



PROCESSES
Contains the name of each database processed through this process-view. Each database whose processing is described in the DESCRIPTION clause of this member must be named in this clause. A separate PROCESSES attribute may specify each database processing requirement in free format.

Note: For IMS processing the PROCESSES IMS CONTAINS attribute should be used to specify the constituent database PCB member names.

PROCESSES IMS CONTAINS
Contains the name of each database PCB referenced through this process-view.

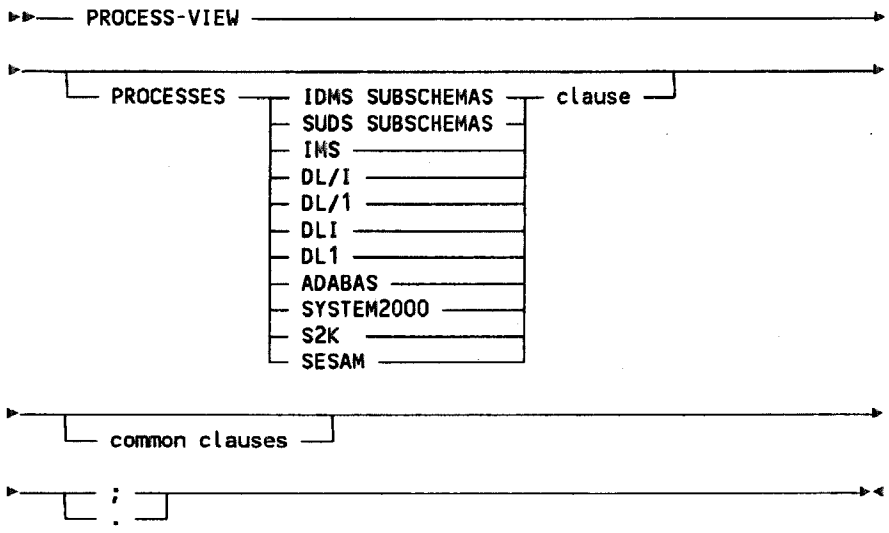
SEGMENT-SEARCH-ARGUMENTS
May contain a list of Segment Search Arguments (SSAa) for sensitive segments contained by the Program Communication Blocks (PCBs) which are referenced by this program.

Multiple USED-IN clauses may be specified where there are many SSAs for a segment.

SSAS
Will only appear for existing PROGRAM members if the keyword SSAS has been specified as an alternative to SEGMENT-SEARCH-ARGUMENTS.

PROCESS-VIEW

Syntax



where

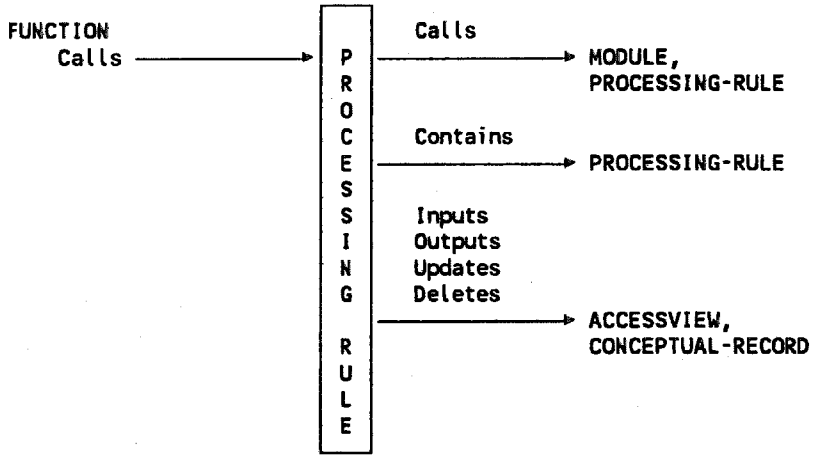
clause is defined in the User Guide specific to your environment.

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

PROCESSING-RULE

A PROCESSING-RULE is a logical process which is designed to be incorporated in more than one function.
The more commonly used relationships to and from PROCESSING-RULE are:



CONTAINS
Contains the name of one or more other common processing rules, which are part of the member being defined.

CALLS
This clause may contain the name of one or more MODULE or PROCESSING-RULE members. A processing rule may call a number of modules, or other common processing rules defined at lower levels.

INPUTS
Contains the names of any ACCESSVIEW and CONCEPTUAL-RECORD members which input information to this member for processing or reference. It should also list the names of members updated by generation of a new version, rather than replacement of an existing version.

PROCESSING-RULE

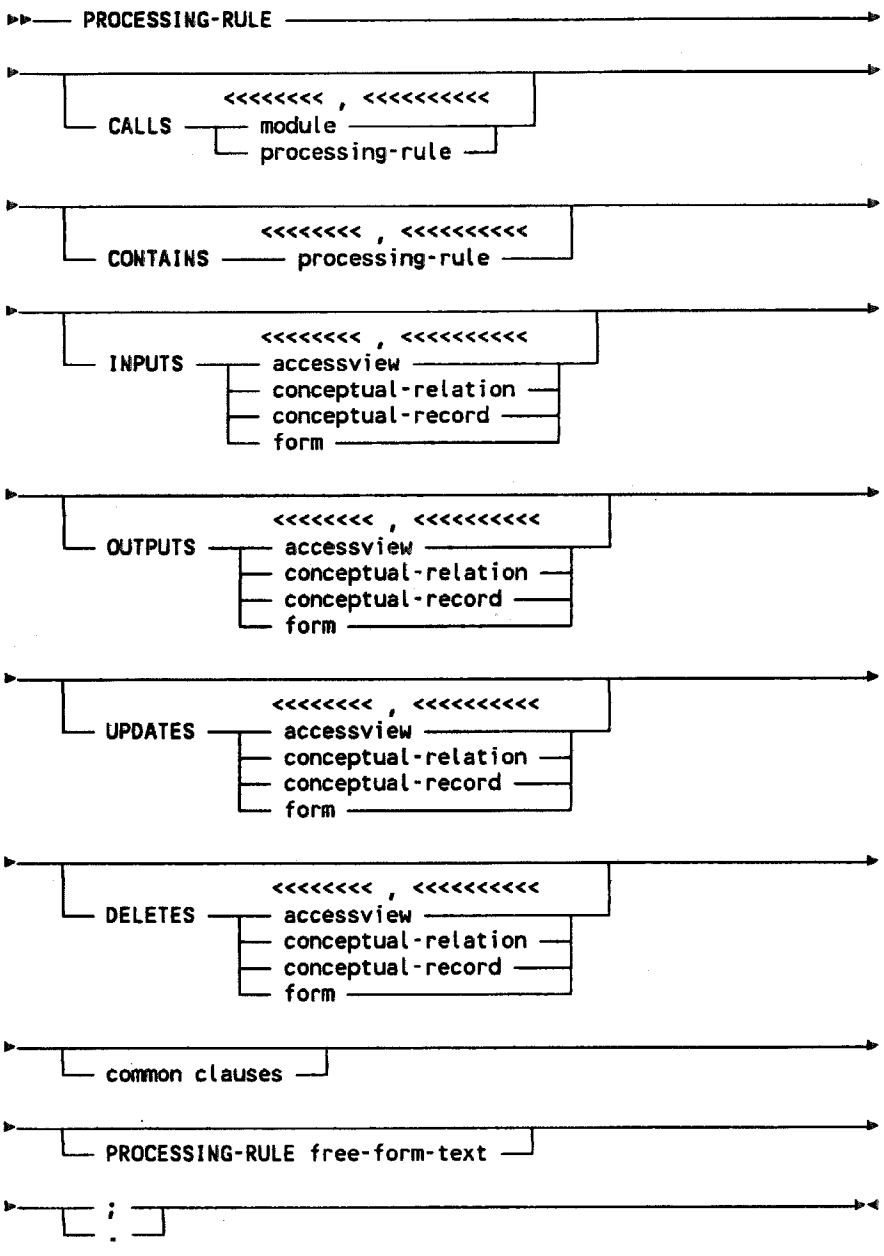
OUTPUTS
Lists the members which receive the information output by the processing rule. It should also list the names of members which this member updates by generating a new version, rather than by replacing an existing version.

UPDATES
The only ACCESSVIEW or CONCEPTUAL-RECORD members named here are those which this member updates by replacing the original version. Members which are updated by generation of a new version should be declared in the INPUTS and OUTPUTS clauses of this member.

DELETES
Contains the name of one or more ACCESSVIEW and CONCEPTUAL-RECORD members. The members named in the clause are deleted when this processing rule executes.

PROCESSING-RULE
Contains free-form text without delimiters, and therefore must be the last clause in the member. It stores a procedural description of what the member does, either in text form, or as pseudocode if a COBOL program is to be generated from it.

Syntax


PROCESSING-RULE
CALLS
module
processing-rule
CONTAINS
processing-rule
INPUTS
accessview
conceptual-relation
conceptual-record
form
OUTPUTS
accessview
conceptual-relation
conceptual-record
form
UPDATES
accessview
conceptual-relation
conceptual-record
form
DELETES
accessview
conceptual-relation
conceptual-record
form
common clauses
PROCESSING-RULE free-form-text
;
.

PROCESSING-RULE

where

module is the name of a MODULE member

processing-rule is the name of a PROCESSING-RULE member

accessview is the name of an ACCESSVIEW member

conceptual-relation is the name of a CONCEPTUAL-RELATION member

form is the name of a FORM member

conceptual-record is the name of a CONCEPTUAL-RECORD member

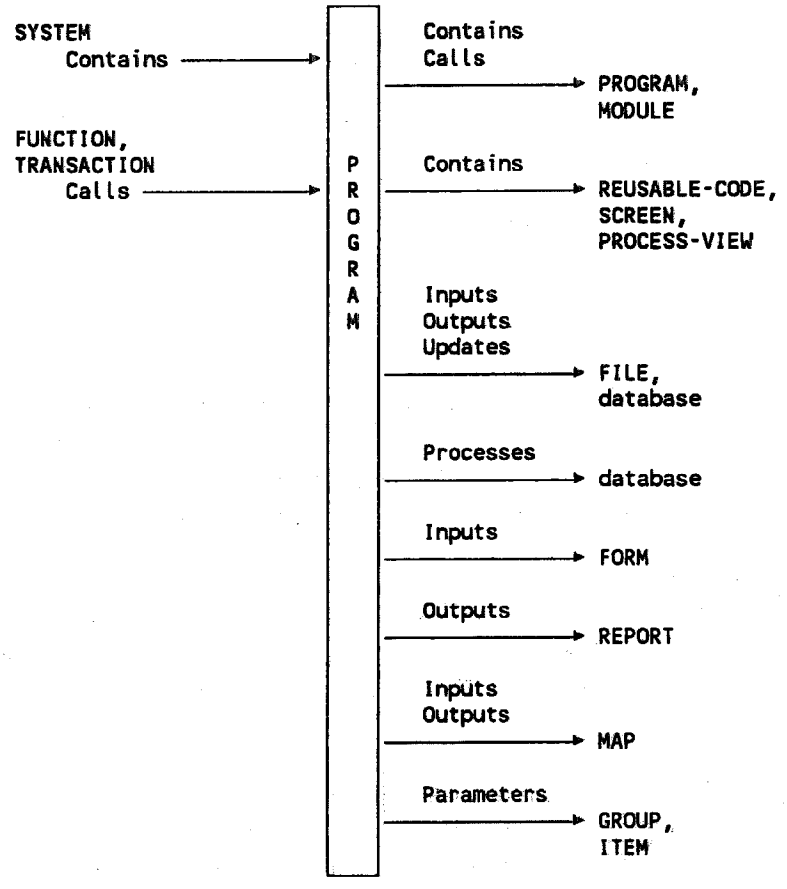
common clauses are any of the clauses common to all member types

Refer to Appendix 2 for details of the common clauses.

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 248 characters.

PROGRAM

A PROGRAM is a set of actions or instructions that a machine is capable of executing as a whole.
The more commonly used relationships to and from PROGRAM are:



CONTAINS

Contains the name of any PROGRAM, MODULE, REUSABLE-CODE, PROCESS-VIEW, REPORT, FILE, SCREEN, GROUP or ITEM members which are used in this member.

PROGRAM

CALLS
This clause may contain the name of one or more PROGRAM and MODULE members. The program is performed as a whole, calling any members named in this clause in the course of execution.

INPUTS
Contains the names of any FILE, GROUP and ITEM members which input information to this member for processing or reference. It should also list the names of members updated by generation of a new version, rather than replacement of an existing version.

OUTPUTS
Lists the members which receive the information output by the program. It should also list the names of members which this member updates by generating a new version, rather than by replacing an existing version.

UPDATES
The only FILE, GROUP or ITEM members named here are those which this program updates by replacing the original version. Members which are updated by generation of a new version should be declared in the INPUTS and OUTPUTS clauses of the program.

PARAMETERS
Contains the names of ITEM, GROUP or FILE members which define parameters passed to this member. It may also include an ENTRY-POINT subclause for each data-name, containing the label of an entry point in this member.

PROCESSES
Contains the name of each database processed through this process-view. Each database whose processing is described in the DESCRIPTION clause of this member must be named in this clause.
A separate PROCESSES attribute may specify each database processing requirement in free format.
NB; For IMS processing the PROCESSES IMS CONTAINS attribute should be used to specify the constituent database PCB member names.

PROCESSES IMS CONTAINS
Contains the name of each database PCB referenced through this process-view.

SEGMENT-SEARCH-ARGUMENTS
May contain a list of Segment Search Arguments (SSAa) for sensitive segments contained by the Program Communication Blocks (PCBs) which are referenced by this program.
Multiple USED-IN clauses may be specified where there are many SSAs for a segment.

SSAS
Will only appear for existing PROGRAM members if the keyword SSAS has been specified as an alternative to SEGMENT-SEARCH-ARGUMENTS.

AUTHOR
Records the name of the author of this member. It is needed if a COBOL source program is generated from this definition. Otherwise the clause is used for general documentation.

LANGUAGE
The name of the computer language in which the member is programmed.

INSTALLATION
Contains the name of the installation where this member is used. It is needed if a COBOL source program is to be generated from this definition. Otherwise the clause is used for general documentation.

DATE-WRITTEN
Contains a date, in your organization's standard format, stating when this member was set up. It is needed if a COBOL source program is generated from the member. Otherwise it is used for general documentation.

SOURCE-COMPUTER
Contains the name of the computer that is the source of data used in this member. This clause is needed if a COBOL source program is to be generated from the definition. Otherwise it is used for general documentation.

PROGRAM

OBJECT-COMPUTER
Contains the type of the computer on which the object module created this program or module will run.
It is needed if a COBOL object program is to beGenerated from this definition. Otherwise it is used for documentation.

SPECIAL-NAMES
Forces the processing code to replace a specific character with another character. This clause is needed if a COBOL object program is to be generated from the member which contains this clause. Otherwise it is used for general documentation.

I-O-CONTROL
Stores any special I-O instructions for the member being defined. It is needed if a COBOL object program is to be generated from this definition. Otherwise the clause is used for general documentation.

ASSIGNMENT
Contains the name of the device assigned to this member. It is needed if a COBOL object program is generated from this definition. Otherwise the clause is used for general documentation.

EDIT-INPUT
Contains instructions about editing input to the member before processing. It is needed if a COBOL source program is to be generated from this member. Otherwise it is used for documentation.

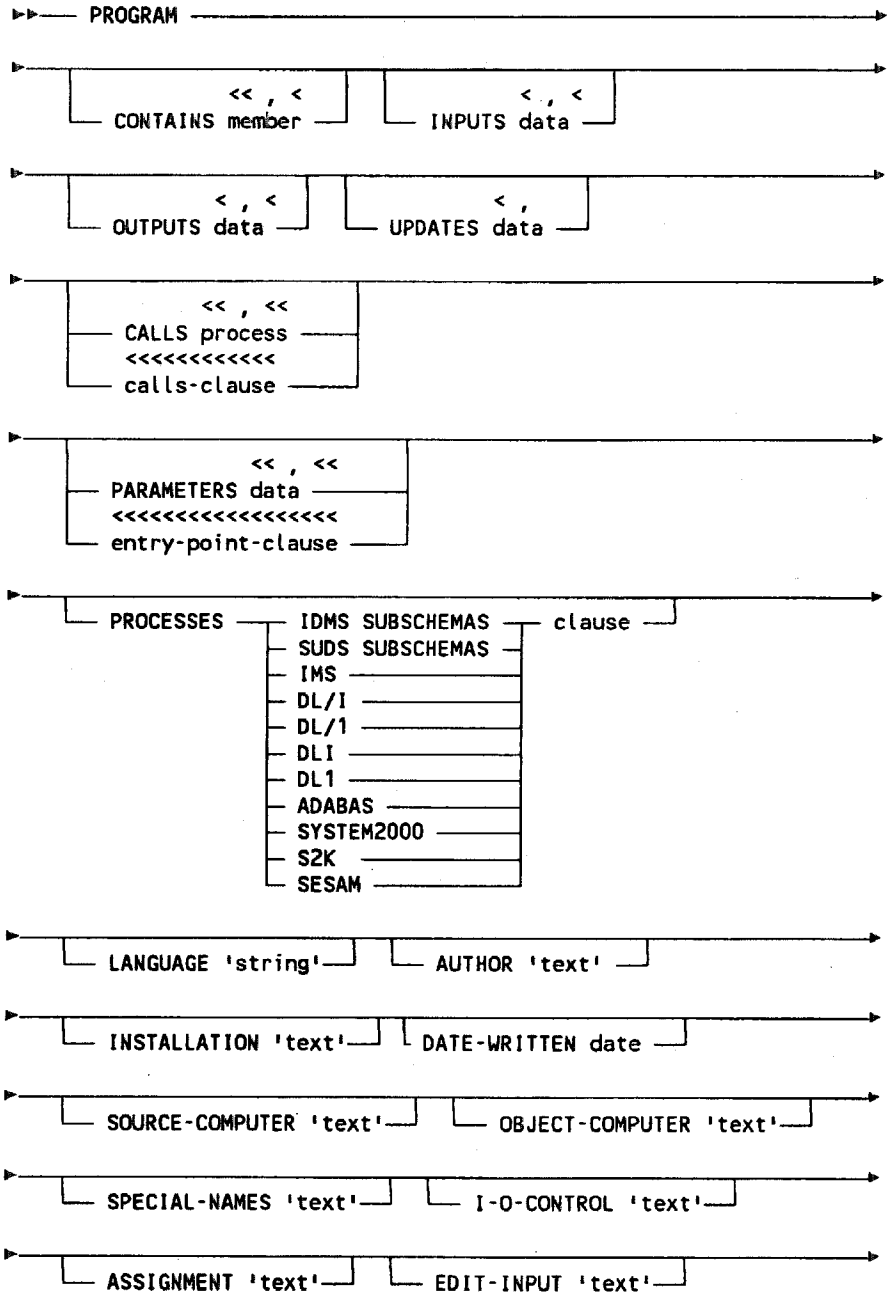
EDIT-OUTPUT
Contains instructions about any editing of output to be done before this member begins processing. It is needed if a COBOL source program is to be generated from this definition. Otherwise the clause is used for general documentation.

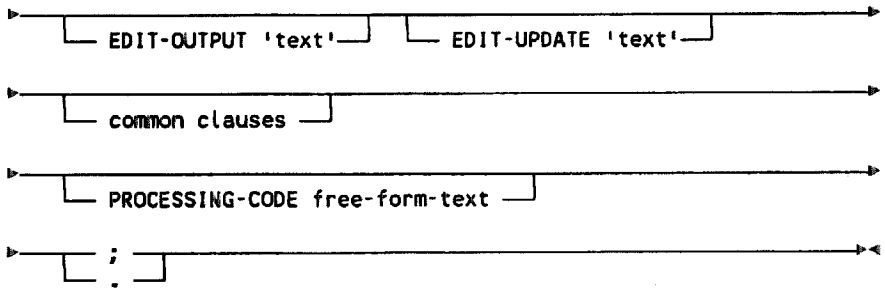
EDIT-UPDATE
Contains instructions about anything which needs editing after being updated by this member. It is needed if a COBOL source program is to be generated from this definition. Otherwise the clause is used for general documentation.

PROCESSING-CODE
Contains processing code as free-form text, and must be the last clause in the member. It is needed if a COBOL source program is to be generated from this definition. Otherwise it is used for general documentation.

PROGRAM

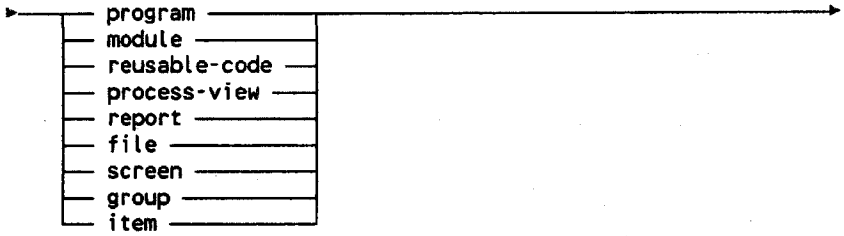
Syntax



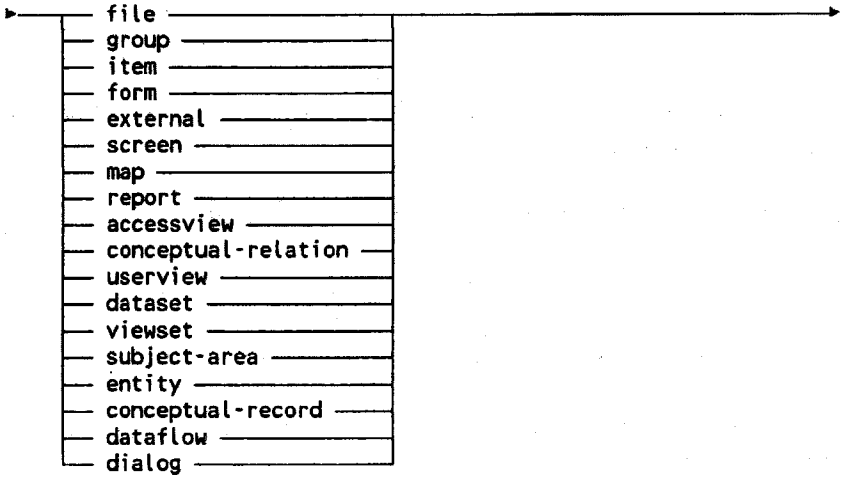


where

member is any of the following:



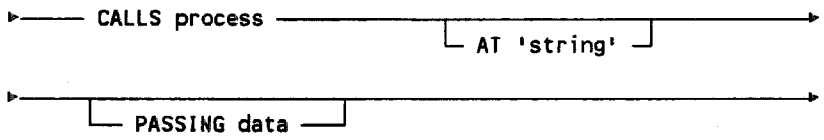
data is any of the following:



process identifies a process member at the same or a lower level in the member type hierarchy

PROGRAM

calls-clause is:



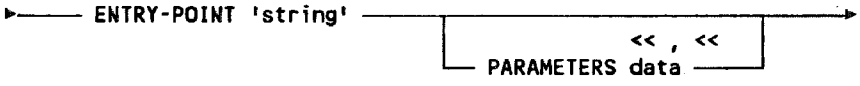
where

process is as defined above

'string' is a character string of not more than 256 characters

data is as defined above

entry-point-clause is:



where **'string'** and **data** are as defined above

clause is defined in the User Guide specific to your environment.

'string' is a character string of not more than 256 characters

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters

date is a date, in the format defined by your installation

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 248 characters.

PROJECT

A PROJECT is the planning and allocation of resources for a period for the accomplishment of a predefined goal.

LIFE-CYCLE
Name of life cycle model under which the project is being run.

RISK-ANALYSIS
Risk analysis of the project.

MIGRATION-DETAILS
Specifies details of the provisions for changeover from old to new systems, such as a schedule of changeover activities.

DATA-PROTECTION-DETAILS
Records the Data Protection measures used by the systems in this project, in accordance with the Data Protection Act. It is needed for anything involving confidential information.

AUDIT-DETAILS
Used by the business analyst, in the course of business design or analysis, to note what the system must be able to produce to fulfil audit requirements.

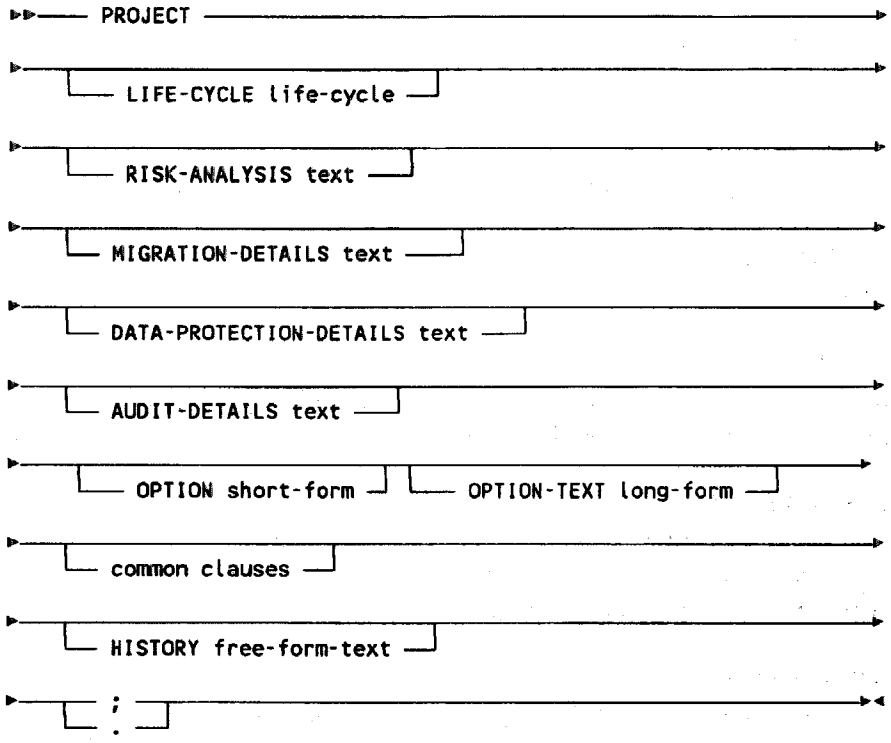
OPTION
Short description of project management components, as shown on the user interface.

OPTION-TEXT
Long description of project management components, as shown on the user interface.

HISTORY
Contains free-form text and must be the last clause in the member. It records details of project validation checks, phase openings and closures. It is automatically updated when the member is accessed.

PROJECT

Syntax



where

life-cycle is the name of a LIFE-CYCLE member

text is a maximum of 32767 delimited character strings, each containing a maximum of 256 characters

short-form is a delimited string of up to 32 characters

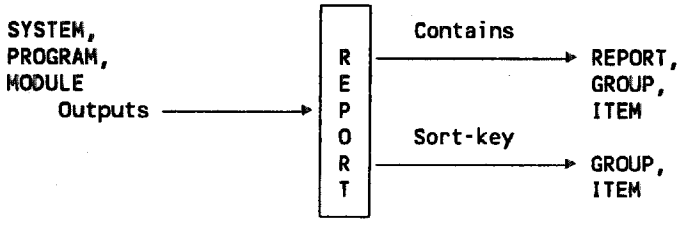
long-form is a delimited string of up to 50 characters

common-clauses are any of the clauses common to all member types. Refer to Appendix 2 for details of the common clauses.

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 256 characters

REPORT

A REPORT is a formal presentation of information, usually in the form of printed output from a computer.
The more commonly used relationships to and from REPORT are:



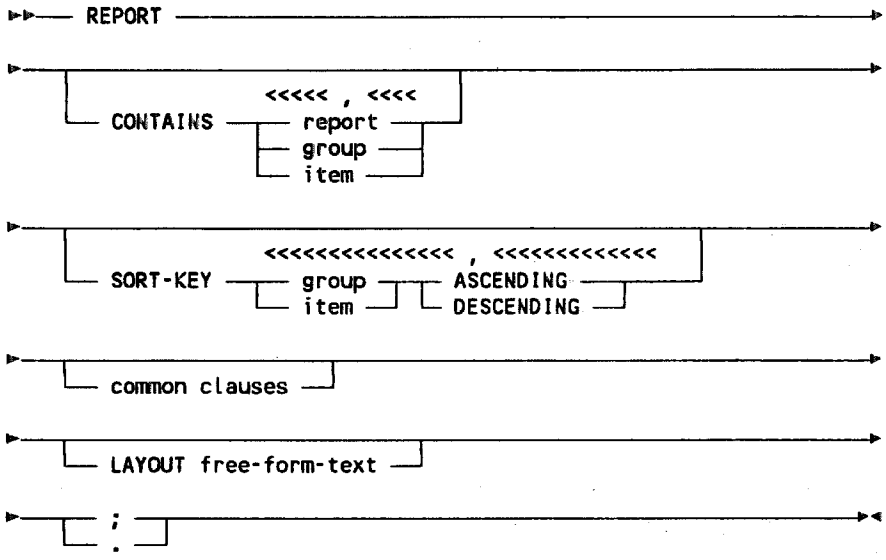
CONTAINS
Contains the name of the REPORT, GROUP or ITEM members which hold the information used in the report.

SORT-KEY
Specifies the component members on which the contents of this member are sorted. For each member named in this clause, the sort sequence (ASCENDING or DESCENDING) must be specified.

LAYOUT
Contains free-form text, and must therefore be the last clause in the member. It is used to establish the exact layout required for the map, such as the positions of protected and unprotected fields.

REPORT

Syntax



where

report is the name of a REPORT member

group is the name of a GROUP member

item is the name of an ITEM member

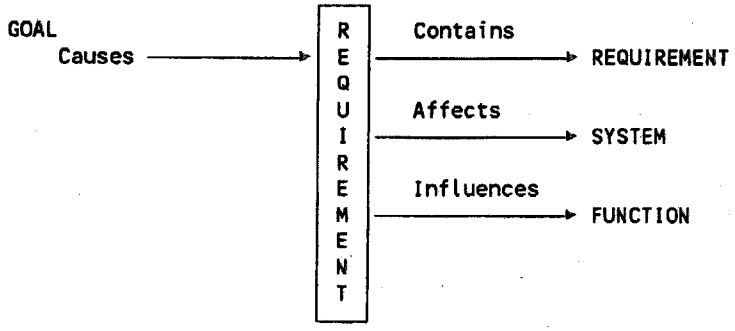
common clauses are any of the clauses common to all member types

Refer to Appendix 2 for details of the common clauses.

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 256 characters

REQUIREMENT

A REQUIREMENT is a statement of an informational need to solve a problem or exploit an opportunity for an organizational unit.
The more commonly used relationships to and from REQUIREMENT are:



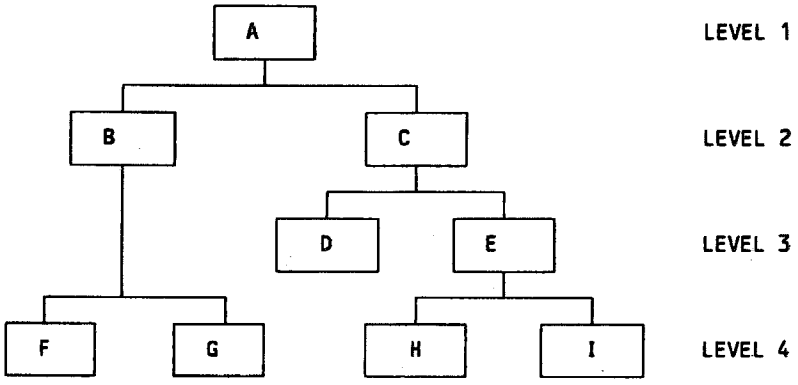
CONTAINS
Contains the name of one or more requirements. The requirements named must be from the same tier as this member, or from a lower tier.

AFFECTS
Contains a description of the ways in which a requirement affects an existing or planned system.

INFLUENCES
Specifies the functions which are influenced by the requirement. Each requirement caused by a goal is likely to influence one or more functions, for example by altering the output from the functions.

LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy.
The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.

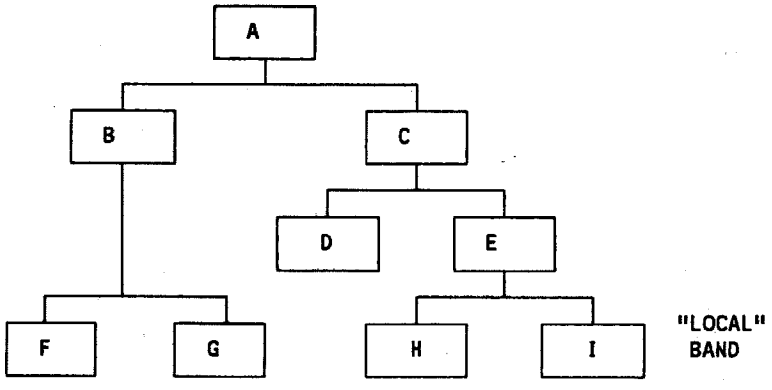
REQUIREMENT



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

BAND

The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.



The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

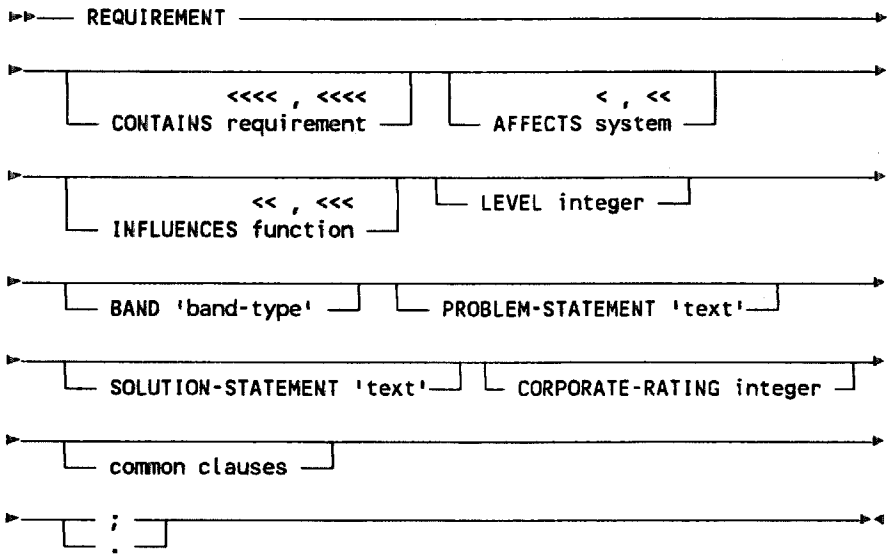
PROBLEM-STATEMENT
Specifies the problem which gives rise to the requirement.

SOLUTION-STATEMENT
Provides a statement of the solution(s) planned to fulfil the requirement.

CORPORATE-RATING
Stores an integer which reflects the value to the enterprise of satisfying the requirement. Any scale of values can be used, but it must be used for every requirement in the enterprise.

REQUIREMENT

Syntax



where

requirement is the name of a REQUIREMENT member

system is the name of a SYSTEM member

function is the name of a FUNCTION member

integer is an integer value of up to 18 digits, optionally preceded by a sign

'band-type' is a text string of up to 78 characters, including delimiters.

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters

Note: the delimiters shown in the above syntax are required when defining a REQUIREMENT member via the command interface.

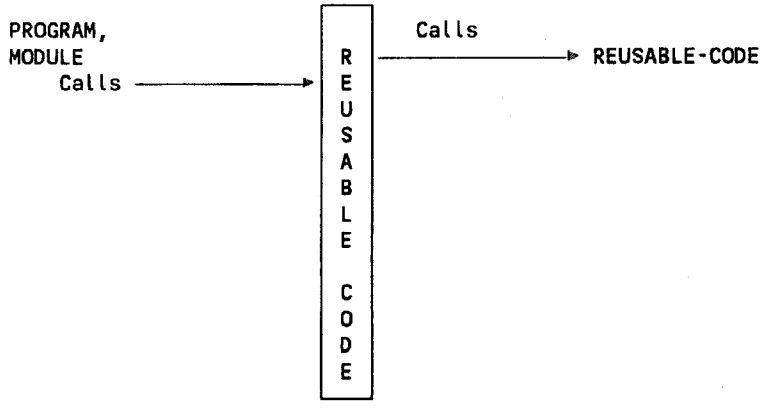
common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

REUSABLE-CODE

REUSABLE-CODE is a piece of program source code which can be incorporated in a module or program at compile time, in a form which may vary depending on the specified values of parameters in the inclusion clause.

The more commonly used relationships to and from REUSABLE-CODE are:



CALLS
This clause may contain the name of one or more REUSABLE-CODE members.

AUTHOR
Records the name of the author of this member. It is needed if a COBOL source program is generated from this definition. Otherwise the clause is used for general documentation.

LANGUAGE
The name of the computer language in which the member is programmed.

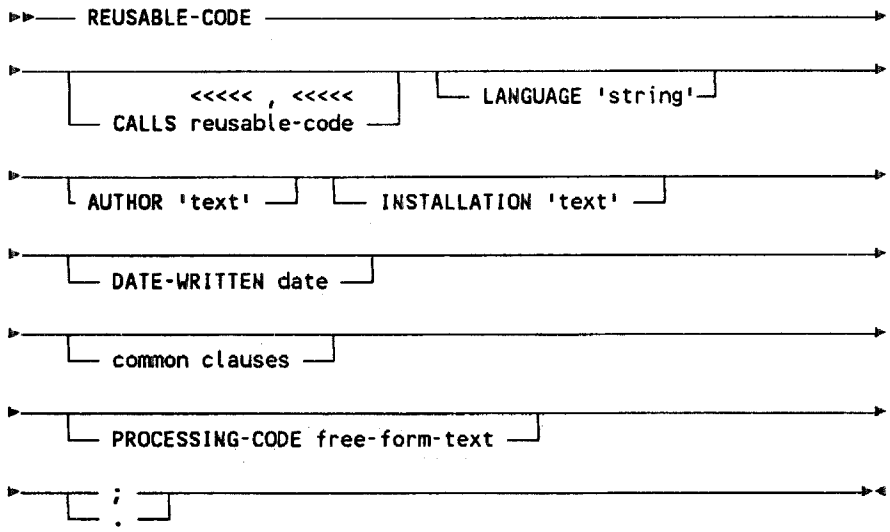
INSTALLATION
Contains the name of the installation where this member is used. It is needed if a COBOL source program is to be generated from this definition. Otherwise the clause is used for general documentation.

REUSABLE-CODE

DATE-WRITTEN
Contains a date, in your organization's standard format, stating when this member was set up. It is needed if a COBOL source program is generated from the member. Otherwise it is used for general documentation.

PROCESSING-CODE
Contains processing code as free-form text, and must be the last clause in the member. It is needed if a COBOL source program is to be generated from this definition. Otherwise it is used for general documentation.

Syntax



where

reusable-code is the name of a REUSABLE-CODE member

'string' is a character string of not more than 256 characters

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters

Note: the delimiters shown in the above syntax are required when defining a REUSABLE-CODE member via the command interface.

date is a date, in the format defined by your installation

common clauses are any of the clauses common to all member types

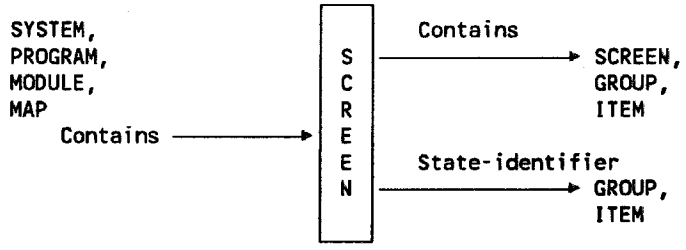
Refer to Appendix 2 for details of the common clauses.

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 248 characters.

SCREEN

A SCREEN is a set of data elements and literals normally comprising exactly one full screen on an interactive terminal display. However, a set of lower level screens can represent areas on a full screen.

The more commonly used relationships to and from SCREEN are:



CONTAINS
Contains the name of each SCREEN, GROUP or ITEM member which holds information needed for this screen. Any SCREEN member named in this clause holds information for an area of this screen.

STATE-IDENTIFIER
Contains the name of the group or item which identifies the state of this member. Possible states are listed in the CONTENTS clause of an item, and the NOTE clause of a group.

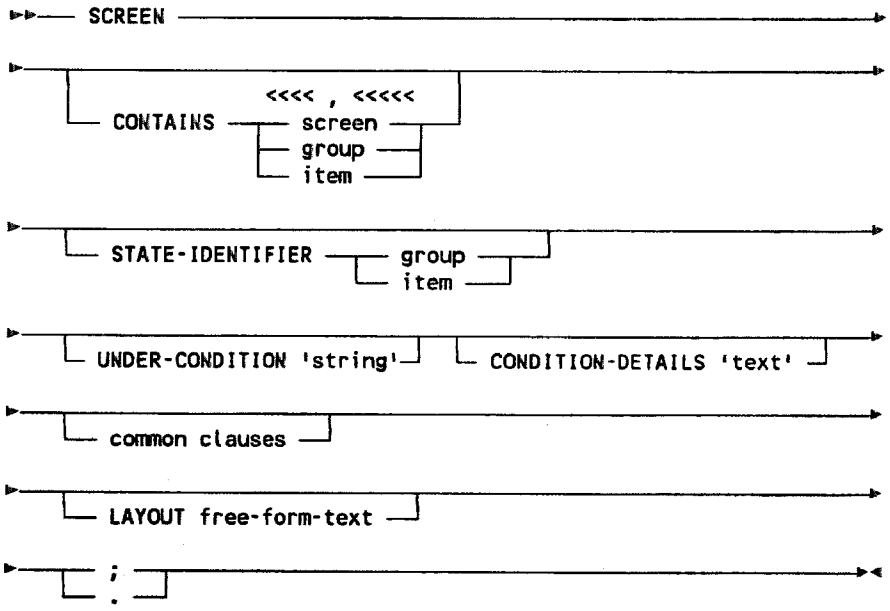
UNDER-CONDITION
Names the conditions which cause a screen to be displayed or changed. If the screen is only displayed by a particular event or set of events, the clause contains a condition expression. If it is always displayed, the clause contains 'ALWAYS'.

CONDITION-DETAILS
This clause provides a full description of the condition, named in the UNDER-CONDITION clause, which causes a screen to be displayed or changed.

LAYOUT
Contains free-form text, and must therefore be the last clause in the member. It is used to establish the exact layout required for the map, such as the positions of protected and unprotected fields.

SCREEN

Syntax



where

screen is the name of a SCREEN member

group is the name of a GROUP member

item is the name of an ITEM member

'string' is a character string of not more than 256 characters

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters

Note: the delimiters shown in the above syntax are required when defining a SCREEN member via the command interface.

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

free-form-text is a maximum of 32767 lines of text, each line containing a maximum of 248 characters.

SESAM

For details of the SESAM interface member types:

- SESAM-STORAGE
- SESAM-TABLE
- SESAM-VIEW

please refer to the SESAM Interface manual (DMR-SESAM).

SQL-DBSPACE

The DBSPACE definition is used to generate ACQUIRE DBSPACE statements.

To document an SQL/DS dbspace in the repository use the SQL-DBSPACE member type. To define the repository member type enter SQL-DBSPACE at the start of your member definition statement. Clauses are available to define the owner (the CREATOR-OWNER clause) and the type of the dbspace. You can define the NHEADER, PAGES, PCTINDEX, PCTFREE, LOCK and STORPOOL parameters in clauses with those names.

Clauses may be declared in any order.

Note: the name you give to an SQL-DBSPACE member may be used as the SQL/DS object name.

AS: Re-Using Existing Member Definitions
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined in another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

CREATOR-OWNER: Defining the Owner of a Dbspace
Use a CREATOR-OWNER clause to specify the owner of a dbspace. The syntax of the clause is as follows:

where **sql-user** is the name of a repository member of the type SQL-USER, which represents the Authorization ID of the owner of the dbspace. The owner of a dbspace is usually the creator, but in SQL Version 2 Release 2, the owner may not be the creator. The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL requirements.

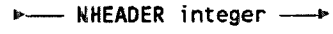
This clause must be present for the successful generation of SQL ACQUIRE and DROP statements.

PUBLIC, PRIVATE: Defining the Type of a Dbspace
Use one of the keywords PUBLIC or PRIVATE to define whether a dbspace is to be a public one or a private one, respectively. The keywords are the same as those used by SQL and they have the same meanings.

The clause is checked on encoding to ensure that only one of these options is specified.

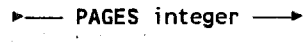
If you omit to specify the type of a dbspace, the SQL default of PRIVATE will apply.

NHEADER: Reserving Space for Header Pages
Use the NHEADER clause to specify the space to be reserved for header pages. The syntax of the clause is as follows:



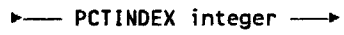
where **integer** is an integer in the range 1-8 inclusive, being the number of logical pages to be reserved. The clause is checked on encoding to ensure that the number of pages you specify is within the permitted range.

PAGES: Defining the Number of Pages Required for a Dbspace
Use a PAGES clause to define the minimum number of logical pages you want a dbspace to have. The syntax of the clause is as follows:



where **integer** is the minimum number of logical pages required.

PCTINDEX: Defining Free Space for Indexes
Use a PCTINDEX clause to define the percentage of free space which you want to be left in a dbspace for indexes. The syntax of the clause is as follows:

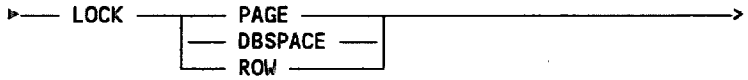


where **integer** is an integer of between 0 and 99. The clause is checked on encoding to ensure that the percentage you specify is within the permitted range.

PCTFREE: Defining Free Space
Use a PCTFREE clause to define the percentage of free space which you want to be left on each page of the dbspace. The syntax of the clause is as follows:

where **integer** is an integer in the range 0 to 99. The clause is checked on encoding to ensure that the percentage you specify is within the permitted range.

LOCK: Defining the Locking Level
Use a LOCK clause to define the locking level for a dbspace.
The syntax of the clause is as follows:



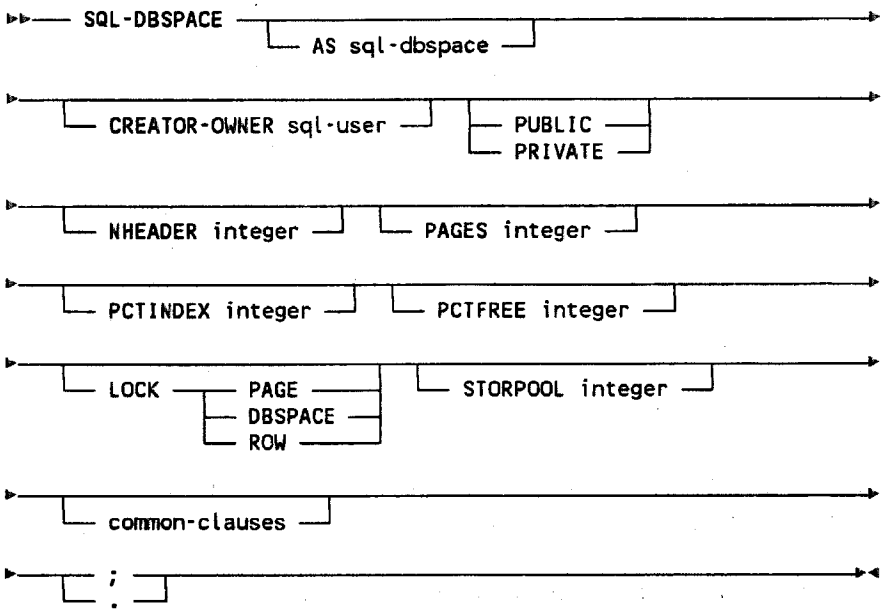
where **PAGE, DBSPACE** and **ROW** are the permitted locking levels. The clause is checked on encoding to ensure that the option specified is a valid one.

Note: You may use the LOCK clause only if you have defined the dbspace as a PUBLIC one.

STORPOOL: Defining the Storage Pool Where Space is to be Acquired
Use a STORPOOL clause to define the number of the storage pool where a dbspace is to be acquired. The syntax of the clause is as follows:

where **integer** is the number of the storage pool.

Syntax



where

sql-dbspace is the name of an SQL-DBSPACE member

sql-user is the name of an SQL-USER member

integer (in the NHEADER clause) is an integer in the range 1 to 8

integer (in the PAGES clause) is an integer within the permitted range, being the number of pages required for the dbspace

integer (in the PCTINDEX clause) is an integer in the range 0 to 99

integer (in the PCTFREE clause) is an integer in the range 0 to 99

integer (in the STORPOOL clause) is an integer within the permitted range, being the number of the storage pool in which the dbspace is to be accquired

common clauses are any of the clauses available to all member types. Refer to Appendix 2 for details of the common clauses.

SQL-INDEX

To document an SQL/DS index in the repository use the SQL-INDEX member type.

Clauses and keywords are available which allow you to specify the owner of an index (the CREATOR-OWNER clause), and the table to be indexed (the ON clause). You can specify the columns which are to be indexed (in a CONTAINS clause). The keyword UNIQUE is available to define an index as unique. You can define the percentage of free space to be left in an index (using the PCTFREE clause).

Clauses may be declared in any order.

The CREATOR-OWNER, ON and CONTAINS clauses must be present for the successful generation of an SQL CREATE statement.

The CREATOR-OWNER clause is the only clause which must be present for the successful generation of an SQL DROP statement.

Note: the name you give to an SQL-INDEX member may be used as the SQL/DS object name.

The naming conventions and the ways in which external names are derived are described in a separate panel.

AS: Re-Using Existing Member Definitions
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

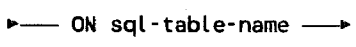
CREATOR-OWNER: Defining the Owner of an SQL-INDEX
Use a CREATOR-OWNER clause to specify the owner of an index. The syntax of the clause is as follows:

where **sql-user** is the name of a repository member of the type SQL-USER, which represents the Authorization ID of the owner of the index. The owner of an index is usually the creator, but in SQL Version 2 Release 2, the owner may not be the creator. The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL requirements.

This clause must be present for the successful generation of SQL CREATE and DROP statements.

UNIQUE: Defining an Index as Unique
Use the keyword UNIQUE to define and index as unique, that is, the column or columns which are being indexed must not have duplicate entries in the table being indexed. If a single column is being indexed, then no value may appear more than once in that column. If more than one column is being indexed, then any given set of values can only appear once.

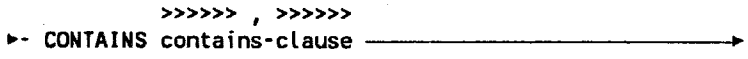
ON: Defining the Table to be Indexed
Use an ON clause to specify the table which is to be indexed.
The format of the clause is as follows:



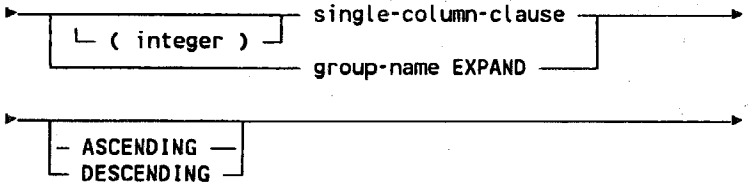
where **sql-table-name** is the name of a repository member of the type SQL-TABLE. The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

This clause must be present for the successful generation of SQL CREATE statements.

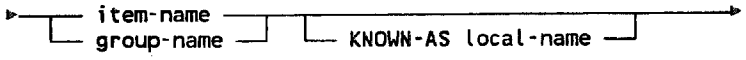
CONTAINS: Defining the Columns being Indexed
Use a CONTAINS clause to specify the column or columns which form the index key. The keyword CONTAINS is followed by one or more column-specifications. The syntax of the clause is as follows:



contains-clause is:



single-column-clause is:



where **(integer)** is the number of columns in a 'column set'. The column specified in the following single-column-clause will be repeated by the number of times you have specified. On generation of an SQL statement each column produced will be suffixed automatically by a number; the first column will be suffixed by 1, the second by 2, and so on. For example, suppose you have a set of two columns, PERIOD_1 and PERIOD_2. If you specify (2) PERIOD in the CONTAINS clause (where PERIOD is the name of the ITEM repository member which represents the column) two columns, PERIOD_1 and PERIOD_2, will be generated.

item-name is the name of a repository member of the type ITEM, which represents one column in the table being indexed.

group-name in the single-column-clause is the name of a repository member of the type GROUP, which represents one column in the table being indexed.

Use a KNOWN-AS clause to specify a local-name which will be used as the name of the SQL/DS indexing column.

Use the group-name EXPAND option to refer to a collection of GROUPs and/or ITEMs, each of which represents a column being indexed. Group-name is the name of a repository member of the type GROUP.

Note: local-names may not be specified for EXPANDed members; this is checked on encoding.

You may specify a maximum of 16 columns in an index key. The clause is checked on generation to ensure that the number of columns specified does not exceed this maximum.

The CONTAINS clause is checked on encoding to ensure that the members specified are of the correct type (that is, GROUPs or ITEMs). The clause is checked on generation to ensure that the length of the derived names of the columns is compatible with the external environment.

The clause is checked on generation also to ensure that no duplicate column-names have been specified.

Each column-name in an index must have the same, corresponding, column-name in the table it indexes.

Use one of the keywords ASCENDING or DESCENDING to determine the sort-sequence for columns in the index. If you omit to specify a sort-sequence, none will be generated and the SQL default will apply.

EXPAND: Defining Several Columns at Once in an SQL-INDEX
Use a group-name EXPAND clause to specify a number of columns at once. The syntax of the clause is as follows:

where **group-name** is the name of a repository member of the type GROUP. The purpose of this clause is to facilitate the generation of several columns in the index at once; it is a shorthand way of referring to a number of GROUPs and ITEMs (that is, all those contained by the specified GROUP), which will each represent one column in the index. Your installation may already have GROUPs defined in its repository which are used in existing applications. These GROUPs may represent records or segments that now need to have counterparts in the SQL/DS environment.

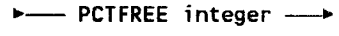
Because of the simpler 'flat-file' structures supported by SQL/DS you need to observe the following points.

A GROUP to be EXPANDed should not contain any ELSE clauses. These give rise to record 'overlays', that is, records in which certain fields may share the same areas of physical storage. In SQL/DS such a concept has no meaning, since a column in an index must have a name unique in the index and cannot 'overlay' or share data with any other column in the index. If you do have an ELSE clause, it will be ignored.

A group to be EXPANDed may contain 'nested' groups as well as items. Nesting can continue to any depth; the only limit is the amount of memory available. However, whereas in segments and host language data structures, such nesting is meaningful, in an SQL/DS index, it is not. Therefore, intermediate levels in the data structure are removed, in order to generate a 'flat' structure.

You may not specify a KNOWN-AS clause or a version for an EXPANDed group; this is checked on encoding. This means that the column-names generated from a group-name EXPAND clause are taken either from the KNOWN-AS names of the ITEMs and/or GROUPs contained by the GROUP specified, or, failing that, from their repository names. The length of any column-name may be no longer than 18 characters; this is checked on encoding. The clause is also checked on generation to ensure that the length of the derived names is compatible with the external environment.

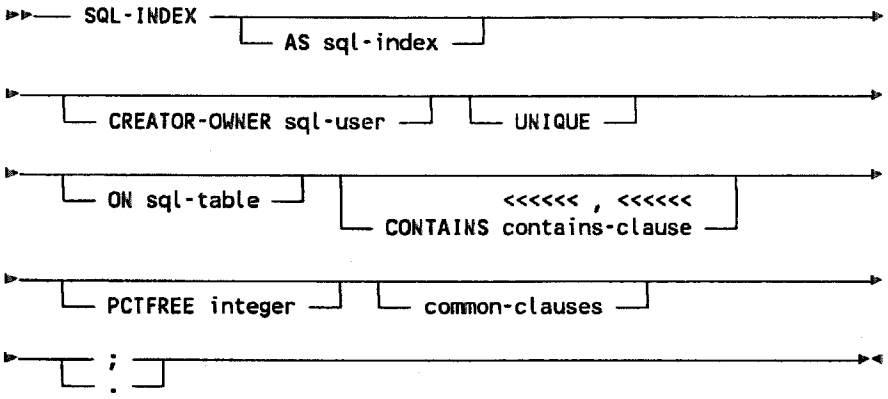
PCTFREE: Defining Free Space
Use a PCTFREE clause to define the percentage of free space you want to be reserved in the index for later updates and insertions. The syntax of the clause is as follows:



where **integer** is an integer in the range 0 to 99, being the percentage of the total space of the index which you want to reserve. The clause is checked on encoding to ensure that the value specified is within the permitted range.

SQL-INDEX

Syntax



where

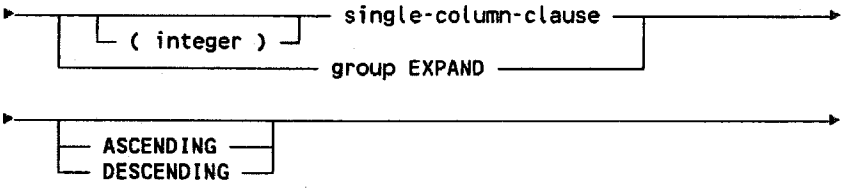
sql-index is the name of an SQL-INDEX member

sql-user is the name of an SQL-USER member

sql-table is the name of an SQL-TABLE member

integer (in the PCTFREE clause) is an integer in the range 0 to 99.

contains-clause is:

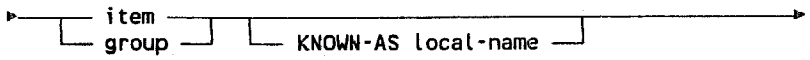


where

integer (in the CONTAINS clause) is the number of columns in a 'column set'

group is the name of a GROUP member

single-column-clause is:



where

item is the name of an ITEM member
group is as defined above
local-name consists of no more than 18 characters

common-clauses are any of the clauses common to all member types

Refer to Appendix 2 for details of the **common-clauses**.

SQL-PRIVILEGE

To document an SQL/DS privilege in the repository use the SQL-PRIVILEGE member type. In the SQL/DS environment you can grant privileges to specific users to allow them to access particular tables, views and programs. You can also grant system authorities to users (in a SYSTEM privilege).

You can record all of these types of privilege in the repository as members of the type SQL-PRIVILEGE, the distinction between them being made by an appropriate keyword in the data definition. You may record only one type of privilege in one member.

Table and view privileges give specified users access to particular SQL/DS objects. For example, an UPDATE privilege on a table gives a user the ability to perform UPDATES on a particular, named, table (defined in the ON clause).

Program privileges allow specified users to run particular programs, and system authority privileges allow you to grant specified users different levels of administrative authority over the SQL/DS environment.

You can record the grantors and recipients of privileges (in GRANTOR and TO clauses respectively).

Table, view and program privileges can be granted with or without the GRANT option (by including or excluding the WITH-GRANT-OPTION keyword). A privilege with a GRANT option is transferable, that is, the recipient may pass on the privilege to another user.

Clauses may be declared in any order.

The privilege-type clause and the TO clause must be present for the successful generation of an SQL GRANT statement.

The definitions recorded in the repository are used to generate SQL GRANT and SQL REVOKE statements. They can also be interrogated using repository interrogation commands to provide database administrators with the ability to analyse the repository model of the SQL/DS security system.

When you generate an SQL REVOKE statement from an SQL-PRIVILEGE member, you must consider carefully whether to remove the member from the repository. You may need to grant the privilege again, in which case you should retain the member in the repository. If you do this, the privileges documented in the repository will reflect your long-term security arrangements and can be used whenever necessary to GRANT and REVOKE SQL privileges.

AS: Re-Using Existing Member Definitions
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

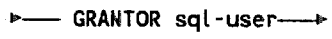
This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

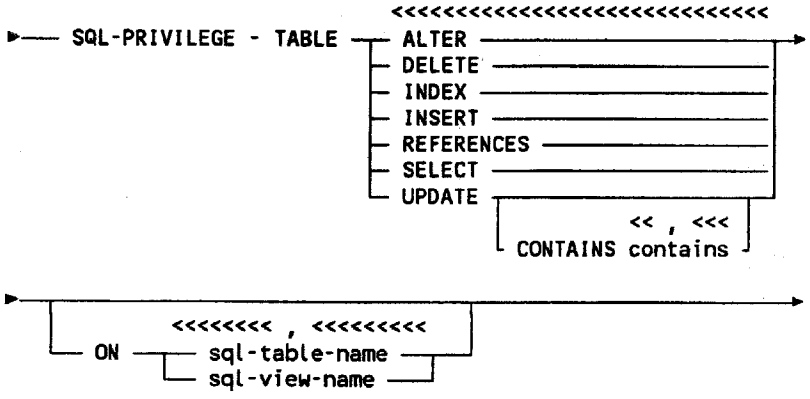
SQL-PRIVILEGE

GRANTOR: Defining the Grantor of an SQL/DS Privilege
Use a GRANTOR clause to define the user who is granting the privilege. The syntax of the clause is as follows:

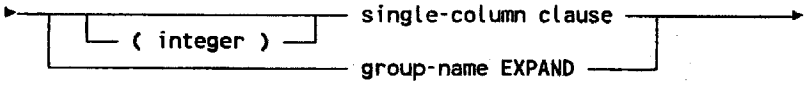


where sql-user is the name of a repository member of the type SQL-USER, which represents the Authorization ID of the user who is granting the privilege. (Typically, the grantor will be a database administrator for a project.) The clause is checked on encoding to ensure that the member specified is of the correct type. This clause is used for documentation purposes only; it is optional and does not affect the generation of an SQL GRANT statement.

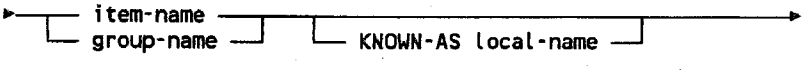
TABLE: Defining Specific Privileges on Tables and Views
Use the keyword TABLE to grant privileges on tables and views. The syntax is as follows:



contains is:



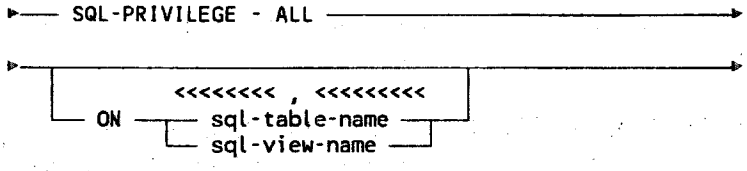
single-column clause is:



The keywords are the same as those used by SQL/DS and they have the same meanings. Use the CONTAINS clause to define the column or columns which the UPDATE privilege applies to. The column(s) thus specified must be defined in the same way as they are in the corresponding clauses of the SQL-TABLE or SQL-VIEW member, which represents the SQL/DS object to which the privilege applies, and they must conform to the same requirements.

sql-table-name and **sql-view-name** are repository members of the appropriate type, which represent the object(s) on which the privilege operates.

ALL: Defining ALL Privileges on Tables and Views
Use the keyword ALL to grant all possible privileges a user may have on a table or view. The syntax is as follows:



where **sql-table-name** and **sql-view-name** are repository members of the appropriate type, which represent the object(s) on which the privilege operates.

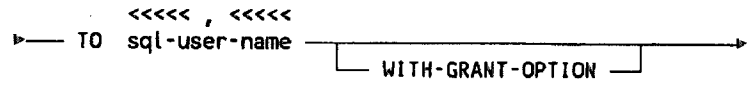
PROGRAM ON: Defining SQL/DS PROGRAM Privileges
Use the PROGRAM ON clause to grant program privileges. The syntax is as follows:

where **program-name** is the name of the PROGRAM which the recipient of the privilege is permitted to run. The PROGRAM must be a repository member and it must have a valid CREATOR-OWNER clause. The SQL-USER specified in the CREATOR-OWNER clause is the original grantor of this PROGRAM privilege.

The clause is checked on encoding to ensure that the privilege specified is a valid one.

TO: Defining the Recipient of SQL/DS Privileges on Tables, Views and Programs

Use a TO clause to define the user to whom a privilege on a table, view or program is being granted. The syntax is as follows:



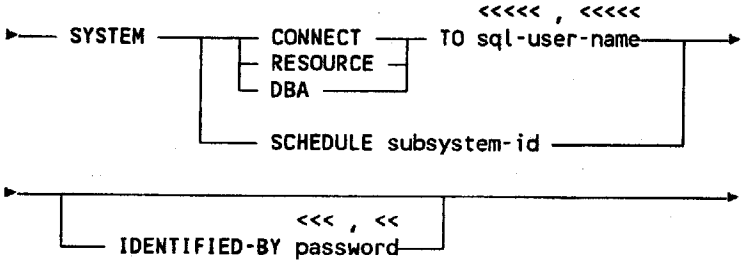
where **sql-user-name** is the name of a member of the type SQL-USER, which represents the Authorization ID of the user to whom the privilege is being granted. The clause is checked on encoding to ensure that the member specified is of the correct type.

Use the optional WITH-GRANT-OPTION keyword to specify that the recipient(s) of the privilege may transfer it to another user.

This clause must be present for the successful generation of an SQL statement.

SYSTEM: Defining SQL/DS SYSTEM Privileges and their Recipients

Use the SYSTEM clause to grant system privileges. The syntax is as follows:



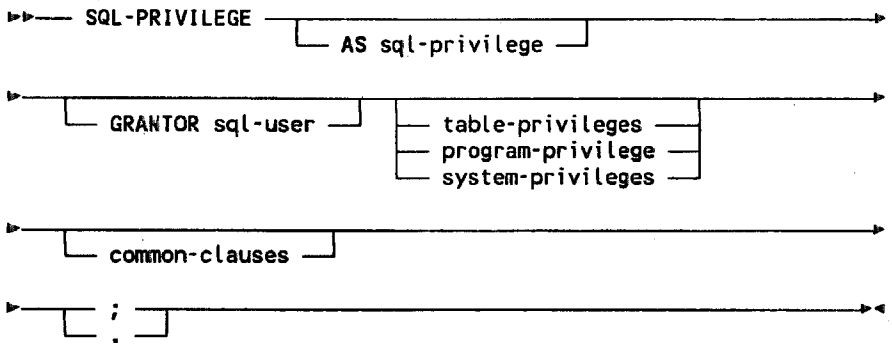
sql-user-name is the name of a repository member of the type SQl-USER, which represents the Authorization ID of the user who is being granted the privilege specified. The clause is checked on encoding to ensure that the member specified is of the correct type.

subsystem-id is the ID of the CICS subsystem running under the VSE guest.

password is a password associated with the SQL/DS user specified, and every user must have an associated password. The subsystem-id may have only one associated password. The password may consist of no more than eight characters; this is checked on encoding. The IDENTIFIED-BY clause may be included if you need to change a password, or specify a new one. The clause must be present for successful generation when a privilege is being granted to a subsystem-id.

SQL-PRIVILEGE

Syntax

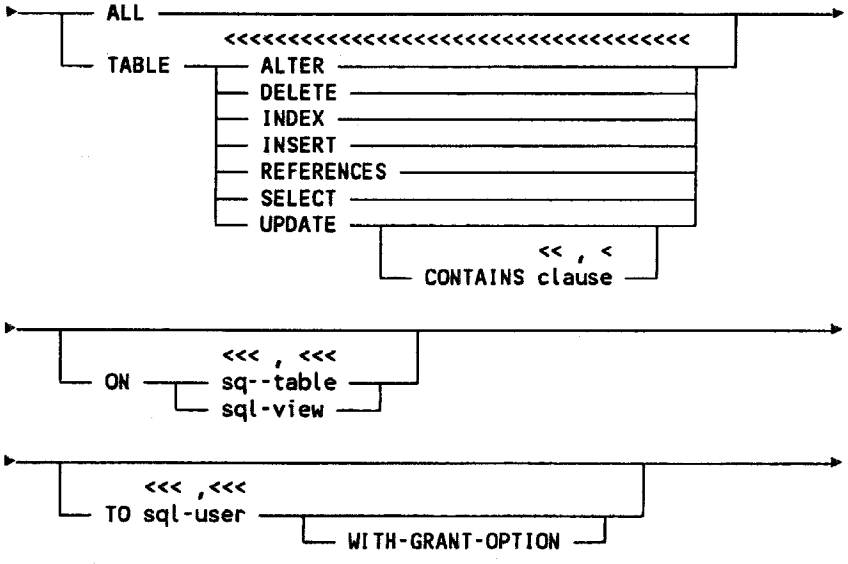


where

sql-privilege is the name of an SQL-PRIVILEGE member

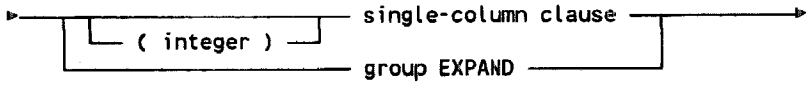
sql-user is the name of an SQL-USER member

table-privileges is:



where

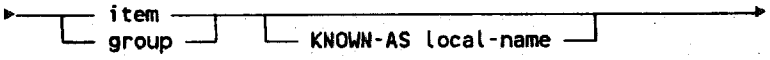
clause is:



where

(integer) is the number of columns in a 'column set'

single-column clause is:

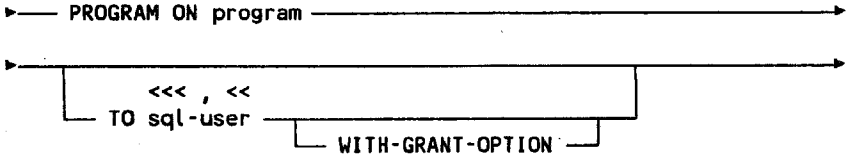


where

item is the name of an ITEM member
group is the name of a GROUP member
local-name consists of no more than eighteen characters

group is as defined above
sql-table is the name of an SQL-TABLE member
sql-view is the name of an SQL-VIEW member
sql-user is the name of an SQL-USER member

program-privilege is:

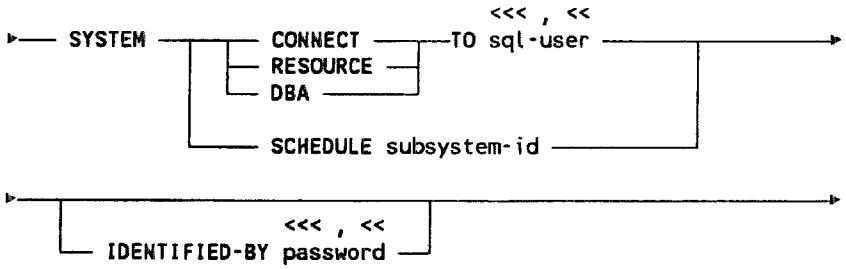


where

program is the name of a PROGRAM member
sql-user is the name of an SQL-USER member

SQL-PRIVILEGE

system-privileges is:



where

sql-user is as defined above
subsystem-id is the name of the subsystem ID of the CICS subsystem
password is an SQL Authorization ID password, consisting of no more than eight characters.

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

SQL-TABLE

To document an SQL/DS table in the repository use the SQL-TABLE member type. Together with SQL-VIEW, it is a member of major interest to the end-user because it contains the user's own data and is not a product of the database administrator or other technical specialist. It is also one of the most used of the SQL/DS member types.

The most important clause available with this member type is the CONTAINS clause. In this clause you specify the ITEM and GROUP members which represent the columns which form the table. In the ITEMs and GROUPs referred to are specified the data types for the columns. The CONTAINS clause, therefore, establishes the relationship between an SQL-TABLE member and ITEM and GROUP members in the repository. In a large repository, these same GROUPs and ITEMs will typically also form part of other file and database segment definitions. For example, installations with IMS may already have GROUP and ITEM definitions in the repository which can now be shared with the SQL/DS environment.

You can also define the owner of a table (using the CREATOR-OWNER clause), and the dbspace in which it is held (using the IN clause). The SQL facilities COMMENT and LABEL are supported by the clauses: SQL-COMMENT and SQL-LABEL.

You will be able to calculate the maximum size of a row in the table and the size of the table (using the SQL SIZE command), if you define a CARDINALITY clause.

You can define and name referential constraints and document the resulting relationships between repository members using the CONSTRAINT clause.

Clauses may be declared in any order, except for the following:

- the definition of a referential constraint (in the CONSTRAINT clause) must follow your definition of the columns which make up the table (in the COLUMNS clause)

- an SQL-COMMENT or SQL-LABEL clause applied to a column must follow the definition for that column (either in a group-name EXPAND clause, or in a single-column-clause). By the same token, an SQL-COMMENT or an SQL-LABEL which applies to a table must not immediately follow a column definition, or it will be taken as a COMMENT or a LABEL on that column.

Note: the name you give to an SQL-TABLE member may be used as the SQL/DS object name.

AS: Re-Using Existing Member Definitions
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

CREATOR-OWNER: Defining the Owner of an SQL/DS Table
Use a CREATOR-OWNER clause to specify the owner of a table. The syntax of the clause is as follows:

where **sql-user** is the name of a repository member of the type SQL-USER, which represents the Authorization ID of the owner of the table. The owner of a table is usually the creator, but in SQL Version 2 Release 2, the owner may not be the creator.

The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

This clause must be present for the successful generation of SQL CREATE, ALTER and DROP statements.

IN: Defining the Dbspace in which a Table is Located
Use an IN clause to specify the dbspace in which a table is located. The syntax of the clause is as follows:

where **sql-dbspace-name** is the name of a repository member of the type SQL-DBSPACE, which represents the dbspace in which the table is to be located.

The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

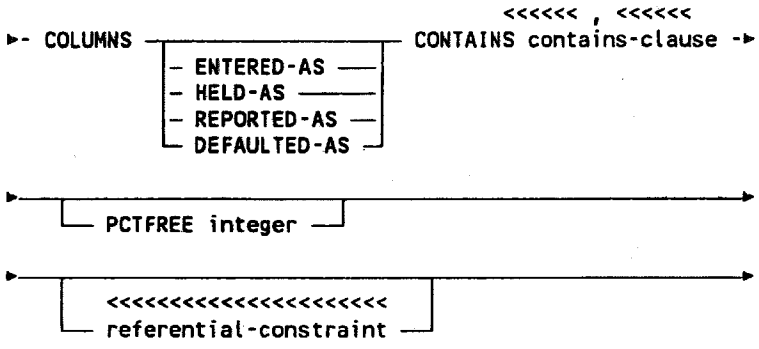
This clause must be present for the successful generation of an SQL CREATE TABLE statement with the SQL CREATE command.

SQL-TABLE

SQL/DS does not require that you specify the dbspace in which a table is located. A table is located in an implicit (default) dbspace if you do not specify an explicit dbspace name in an SQL CREATE TABLE statement.

However, it is good data administration practice to document in the repository all SQL/DS objects, (including implicit objects), and their relationships to one another. MSP's support for SQL generation therefore requires that an IN clause is specified in a SQL-TABLE member definition.

COLUMNS: Defining the Columns which Make up an SQL/DS Table
Use a COLUMNS clause to specify the columns of a table. The syntax of the clause is as follows:



ENTERED-AS, HELD-AS, REPORTED-AS and DEFAULTED-AS are the form keywords. One of them may be used to specify the form of all the ITEMs and/or GROUPs which represent the columns of the table. If none is specified, then the DEFAULTED-AS form of the ITEMs and/or GROUPS is used. If any ITEM or GROUP has no DEFAULTED-AS form, then the usual DATAMANAGER defaults apply.

The purpose of the form keyword is to allow the generation of correct data types for host language data structures. Therefore, although this clause is not mandatory for generation, you are strongly advised to include it when you define a table in the repository, in order to ensure that the correct data types are generated in SQL PRODUCE statements or host language data structures.

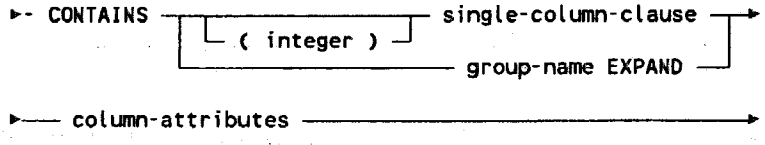
Use the CONTAINS clause to specify the ITEMs or GROUPs which represent the columns of a table.

The PCTFREE parameter is the percentage of space in each index page reserved for later additions or amendments to the primary key. The clause is checked on encoding to ensure that the value specified is within the permitted range. A PCTFREE clause is required only if you define a primary key for a table.

Use the referential-constraint clause to specify any referential constraint(s) which the table may have.

CONTAINS: Defining the ITEMs and/or GROUPs which Represent Columns'

Use a CONTAINS clause to specify the ITEMS and/or GROUPs which represent the columns which form the table, and to specify the attributes of these columns. The syntax of the clause is as follows:



where **(integer)** is the number of columns in a 'column set'. The column specified in the following single-column-clause will be repeated by the number of times you have specified.

On generation of an SQL statement each column produced will be suffixed automatically by a number; the first column will be suffixed by 1, the second by 2, and so on. For example, suppose you have a set of two columns, PERIOD_1 and PERIOD_2. If you specify (2) PERIOD in the CONTAINS clause (where PERIOD is the name of the ITEM repository member which represents the column) two columns, PERIOD_1 and PERIOD_2, will be generated.

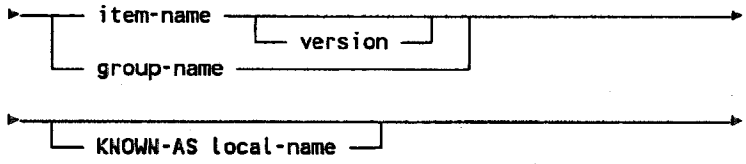
Use a single-column-clause to specify the name of an ITEM or GROUP repository member; each ITEM or GROUP (that is, the ITEMs and/or GROUPs contained by it) represents one column.

Use the group-name EXPAND clause to specify a GROUP member, which represents one or more columns.

A table may have up to 255 generated columns; this figure includes all columns generated as a result of using the "group-name EXPAND" option or the 'column set' option. The clause is checked on generation to ensure that the number of columns specified does not exceed the maximum number permitted.

Use the column-attributes clauses to specify attributes which apply to all columns generated from the preceding clause.

Defining a Single Column in an SQL-TABLE
Use a single-column clause to specify an ITEM or a GROUP repository member which represents a single column. The syntax of the clause is as follows:



item-name is the name of a repository member of the type ITEM; version is the version number of the named ITEM. If no version is specified, then version 1 is assumed by default.

group-name is the name of a repository member of the type GROUP. The GROUP(s) and/or ITEMS(s) contained by this GROUP are combined to represent one column in the table.

The clause is checked on encoding to ensure that the member specified is of one of the correct types (either a GROUP or an ITEM).

Use a KNOWN-AS optional clause to specify a local-name for the ITEM or GROUP. The **local-name** may be no longer than 18 characters; this is checked on encoding and on generation. This local-name is the name of the column. If you do not specify a local-name in this clause, an alias specified in the ITEM or GROUP will be used as the column-name, (assuming your environment is tailored to generate aliases as external names)

If no alias is specified, then the ITEM or GROUP member-name will be used as the column-name (reduced, if necessary, to 18 characters by the name reduction process).

Duplicate column names are not permitted. The clause is checked on generation to ensure that no duplicate column-names are present.

EXPAND: Defining Several Columns at Once in an SQL-TABLE
Use a group-name EXPAND clause to specify a number of columns at once. The syntax of the clause is as follows:

where **group-name** is the name of a repository member of the type GROUP. The purpose of this clause is to facilitate the generation of several columns in the table at once; it is a shorthand way of referring to a number of GROUPs and ITEMs (that is, all those contained by the specified GROUP), which will each represent one column in the table. Your installation may already have GROUPs defined in its repository which are used in existing applications. These GROUPs may represent records or segments that now need to have counterparts in the SQL environment.

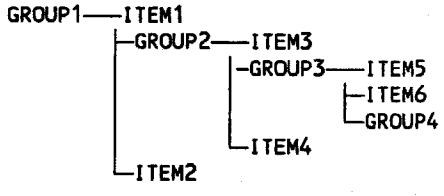
Because of the simpler 'flat-file' structures supported by SQL/DS you need to observe the following points.

A GROUP to be EXPANDed should not contain any ELSE clauses. These give rise to record 'overlays', that is, records in which certain fields may share the same areas of physical storage. In SQL/DS such a concept has no meaning, since a column in a table must have a name unique in the table and cannot 'overlay' or share data with any other column in the table. If you do have an ELSE clause, it will be ignored.

A GROUP to be EXPANDed may contain 'nested' GROUPs as well as ITEMs. Nesting can continue to any depth; the only limit is the amount of memory available. However, whereas in segments and host language data structures, such nesting is meaningful, in an SQL/DS table, it is not. Therefore, intermediate levels in the data structure are removed, in order to generate a 'flat' structure.

You may not specify a KNOWN-AS clause or a version for an EXPANDed group; this is checked on encoding. This means that the column-names generated from a group-name EXPAND clause are taken either from the KNOWN-AS names of the ITEMs and/or GROUPs contained by the GROUP specified, or, failing that, from their repository names. The length of any column-name may be no longer than 18 characters; this is checked on encoding. The clause is also checked on generation to ensure that the length of the derived names is compatible with the external environment.

Example of a Nested GROUP Structure
Consider the following nested structure which is shown in diagram form for clarity -



Such a structure would be represented in PL/1 as -

```
01 GROUP1,
  02 ITEM1,
  02 GROUP2,
     03 ITEM3
     03 GROUP3
        04 ITEM5
        04 ITEM6
        04 GROUP4
     03 ITEM4
  02 ITEM2
```

In SQL/DS, 'higher' level groups GROUP1, GROUP2 and GROUP3 would be removed to produce a 'flat' one-level structure -

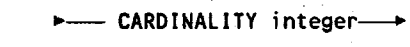
```
01 ITEM1
01 ITEM3
01 ITEM5
01 ITEM6
01 GROUP4
01 ITEM4
01 ITEM2
```

Note: GROUP4, which does not have any lower level, would be considered to be an elementary field; its data type defaults to CHAR(1).

If you consider the original nested structure as a 'tree' and a leaf of the tree as a field which does not have any lower levels, then the root of the tree is taken as the first level and only the 'leaves' of the tree are taken as the second level.

CARDINALITY: Specifying an Estimate of the Number of Rows in an SQL/DS Table

Use the CARDINALITY clause to specify an estimate of the number of rows which will be contained in the table. The syntax of the clause is as follows:



where **integer** is the number of rows which you foresee the table will contain. The integer may be no greater than 2147483647.

This clause must be present if you wish to estimate the total size of the table using the SQL SIZE command.

SQL-COMMENT: Defining an SQL-COMMENT on a Table or a Column

Use an SQL-COMMENT clause to specify that a table or column is to have an associated SQL comment. The syntax of the clause is as follows:

where "**string**" is the comment. The comment must be a character string consisting of no more than 254 characters, enclosed in quotes. The length of the string is checked on encoding.

If you use an SQL-COMMENT clause to qualify a column, it must follow the contains-clause which defines the column.

This clause must be present if you want to generate SQL COMMENT ON statements from an SQL-TABLE member definition.

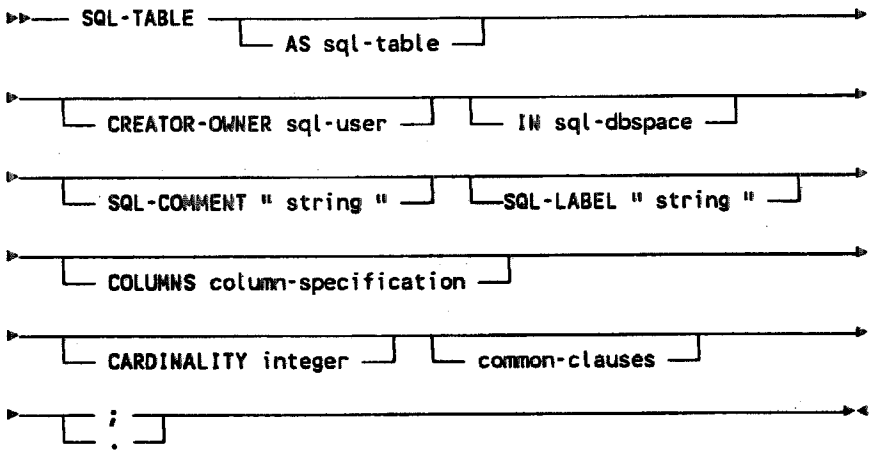
SQL-LABEL: Defining an SQL-LABEL on a Table or a Column
Use an SQL-LABEL clause to specify that a table or column is to have an associated SQL LABEL. The syntax of the clause is as follows:

where "**string**" is the label. The label must be a character string consisting of no more than 30 characters, enclosed in quotes. The length of the string is checked on encoding.

If you use the SQL-LABEL clause to qualify a column, it must follow the contains-clause which defines the column.

This clause must be present if you want to generate SQL LABEL ON statements from an SQL-TABLE member definition.

Syntax



where

sql-table is the name of an SQL-TABLE member

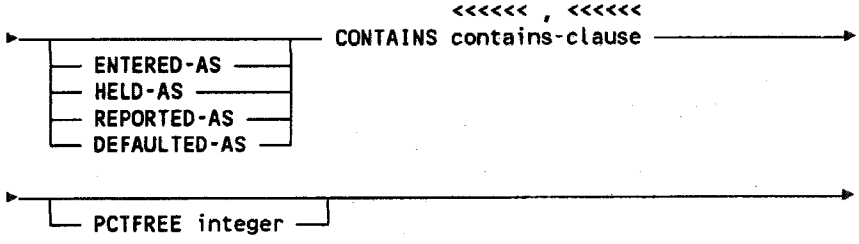
sql-user is the name of an SQL-USER member

sql-dbspace is the name of an SQL-DBSPACE member

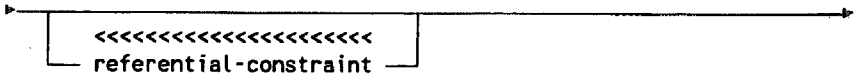
string (in the SQL-COMMENT clause) is a character string of no more than 254 characters

string (in the SQL-LABEL clause) is a character string of no more than 30 characters

column-specification is:

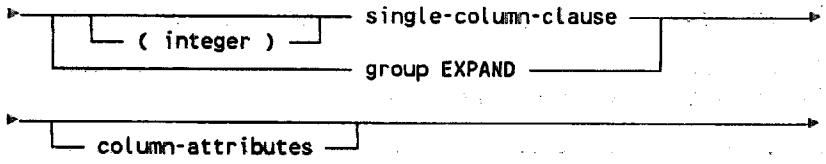


SQL-TABLE



where

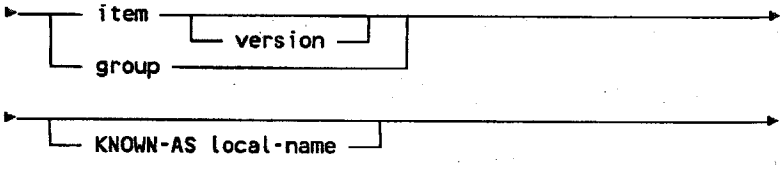
contains-clause is:



where

integer (in the CONTAINS clause) is the number of columns in a 'column set'

single-column-clause is:

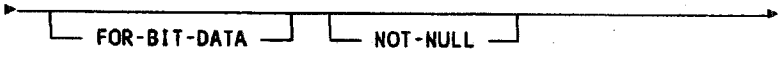


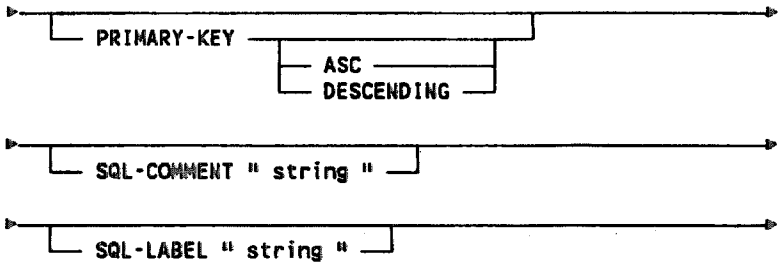
where

item is the name of an ITEM member
version is an integer in the range 1 to 15
group is the name of a GROUP member
local-name consists of no more than 18 characters

group is as defined above

column-attributes is:



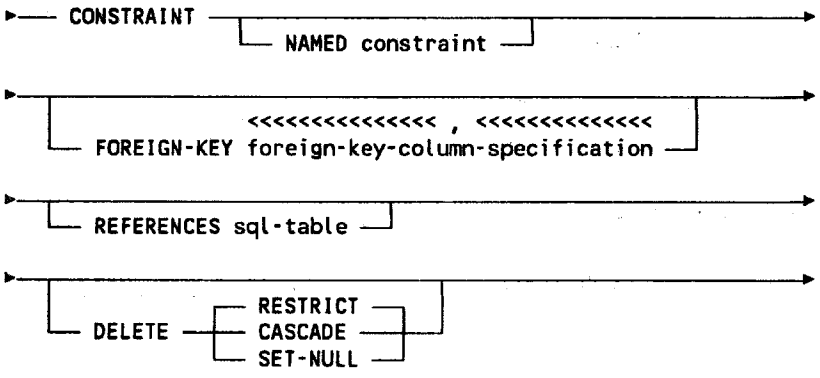


where

string (in the SQL-COMMENT clause) is a character **string** of no more than 254 characters
string (in the SQL-LABEL clause) is a character **string** of no more than 30 characters

integer (in the PCTFREE clause) is the percentage of free space in each index page

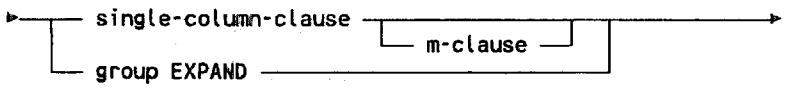
referential-constraint is:



where

constraint is the name of the constraint, consisting of no more than 18 characters

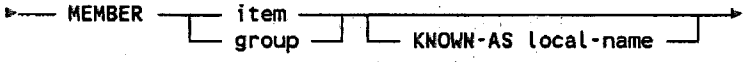
foreign-key-column-specification is:



where

single-column-clause is as above

m-clause is:



where

item and **group** are as defined above
local-name consists of no more than 18 characters

group is as defined above

sql-table is the name of an SQL-TABLE member

integer (in the CARDINALITY clause) is the number of rows which you estimate the table will have

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

SQL-USER

To document an SQL/DS Authorization ID in the repository use the SQL-USER member type.

Use an SQL-USER member to document the owner or creator of SQL/DS objects. Normally, the creator is also the owner, but if you have SQL/DS Version 2 Release 2, it is possible for the owner to be a user other than the creator. This allows one user to create objects on behalf of another user who is the owner.

The SQL/DS object types which are associated with creators and/or owners are dbspaces, tables, views and indexes. The repository member types SQL-TABLE, SQL-VIEW and SQL-INDEX have a CREATOR-OWNER clause which must be present for the successful generation of an SQL statement. The clause is labelled CREATOR-OWNER because it is used to generate the prefix to the SQL/DS name of the table, view or index (to give the fully qualified SQL/DS name). The prefix can be the name either of the creator, or of the owner (if this differs from the Authorization ID of the signed-on user).

SQL-USER members are specified as the recipients of privileges in SQL-PRIVILEGE members.

The SQL-USER member type may contain definitions of synonyms (in the SYNONYMS clause). SQL CREATE SYNONYM statements are generated from this clause.

AS: Re-Using Existing Member Definitions
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

```
AS member
```

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

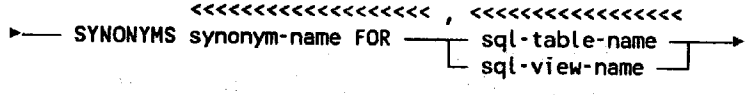
- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

SYNONYMS: Defining SQL/DS Synonyms for Tables and Views
Use a SYNONYMS clause to define synonyms for tables and views owned by the user whose Authorization ID you are documenting. (The user may be yourself, or another user.) The synonyms defined here are used to generate SQL CREATE SYNONYM statements. The syntax of the clause is as follows:

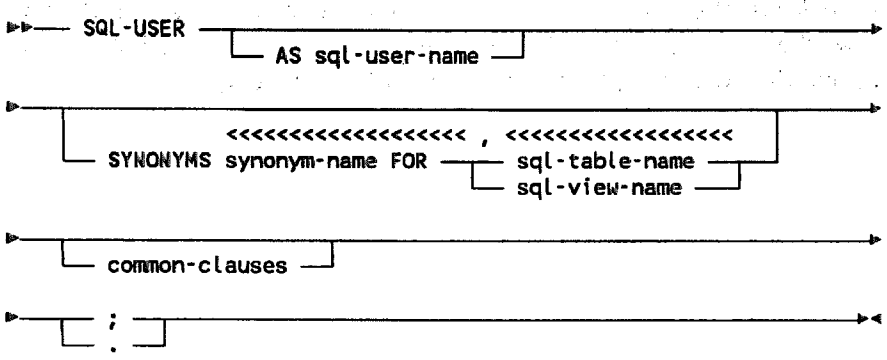


where **synonym-name** is the synonym to be created for the table or view. The synonym may be no longer than 18 characters. The SQL/DS synonym for the object is taken directly from the name you specify in this clause. The length of the name is checked on encoding.

sql-table-name and **sql-view-name** are the names of SQL-TABLE and SQL-VIEW members respectively, one of which represents the object which is to have the synonym specified. The clause is checked on encoding to ensure that the member specified is of one of the correct types.

SQL-USER

Syntax



where

sql-user-name is the name of an SQL-USER member

synonym-name is a synonym, consisting of no more than 18 characters

sql-table-name is the name of an SQL-TABLE member

sql-view-name is the name of an SQL-VIEW member

common clauses are any of the clauses available to all member types.

Refer to Appendix 2 for details of the common clauses.

SQL-VIEW

To document an SQL/DS view in the repository use the SQL-VIEW member type. SQL/DS views are possibly the SQL/DS objects most often used by end users, therefore the SQL-VIEW member type is, with SQL-TABLE, of major importance.

The SQL-VIEW member type bears many similarities to SQL-TABLE. The main similarity is that the columns which form the view are defined in a column-specification clause, in which you specify the ITEM and GROUP members which represent the columns which form the view. As with SQL-TABLE, therefore, this clause establishes relationships between an SQL-VIEW member and ITEM and GROUP members in the repository. Clauses which are particular to the SQL-VIEW member-type (the SELECT, FROM, WHERE and HAVING clauses, which define a view subselect) also establish relationships between a view and the base table(s) or view(s) which the columns, that form the view, belong to.

You can define the owner of a view, by using the CREATOR-OWNER clause. The SQL/DS functions COMMENT and LABEL are supported in an SQL-VIEW member, as in an SQL-TABLE member, by the SQL-COMMENT and SQL-LABEL clauses.

The SQL subselect part of the SQL statement which is generated from an SQL CREATE VIEW command is supported by the optional SELECT clause and its dependent FROM, WHERE and HAVING clauses.

You may declare the clauses in any order, except that the column-attributes which qualify a particular column must be declared immediately following the definition of that column. This is especially important if you use SQL-COMMENT and SQL-LABEL clauses to qualify columns. If they do not appear immediately after the column-specification of the column to which they apply, they will be taken as applying to the whole view.

Note: the name you give to an SQL-VIEW member may be used as the SQL/DS object name.

AS: Re-Using Existing Member Definitions
You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and re-use its clauses.

To re-use clauses already defined an another repository member, enter:

AS member

member is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW.

This enables you to:

- make use of existing member definitions
- avoid re-keying data
- save space in the repository
- share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

CREATOR-OWNER: Defining the Owner of an SQL/DS View
Use a CREATOR-OWNER clause to specify the owner of a view.
The syntax of the clause is as follows:

where sql-user is the name of a repository member of the type SQL-USER, which represents the Authorization ID of the owner of the view. The owner of a view is usually the creator, but in SQL Version 2 Release 2, the owner may not be the creator.

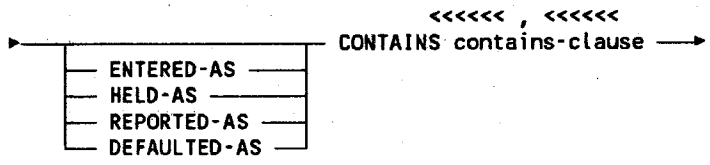
The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

This clause must be present for the successful generation of SQL CREATE and DROP statements.

ENTERED-AS, HELD-AS, REPORTED-AS, DEFAULTED-AS: Defining Columns for an SQL-VIEW

Use a column-specification clause to specify the columns of a view. The column-specification clause is equivalent to the COLUMNS clause in an SQL-TABLE definition.

The syntax of the clause is as follows:



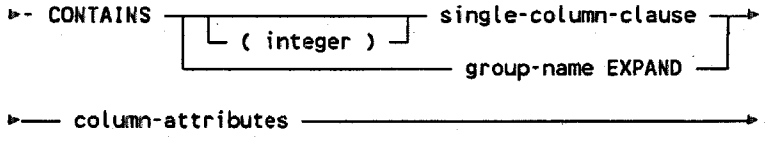
ENTERED-AS, HELD-AS, REPORTED-AS and DEFAULTED-AS are the form keywords. One of them may be used to specify the form of all the ITEMS and/or GROUPs which represent the columns of the view. If none is specified, then the DEFAULTED-AS form of the ITEMs and/or GROUPs is used. If any ITEM or GROUP has no DEFAULTED-AS form, then the usual DATAMANAGER defaults apply.

The purpose of the form keyword is to allow the generation of correct data types for host language data structures. Therefore, although this clause is not mandatory for generation, you are strongly advised to include it when you define a view in the repository, in order to ensure that the correct data types are generated by SQL PRODUCE statements for host language data structures.

Use the CONTAINS clause to specify the ITEMs or GROUPs which represent the columns of a view.

CONTAINS: Defining the ITEMs and/or GROUPs which Represent Columns in an SQL/DS View

Use a CONTAINS clause to specify the ITEMs and/or GROUPs which represent the columns which form the view, and to specify the attributes of these columns. The syntax of the clause is as follows:



where **(integer)** is the number of columns in a 'column set'. The column specified in the following single-column-clause will be repeated by the number of times you have specified. On generation of an SQL statement each column produced will be suffixed automatically by a number; the first column will be suffixed by 1, the second by 2, and so on. For example, suppose you have a set of two columns, PERIOD_1 and PERIOD_2. If you specify (2) PERIOD in the CONTAINS clause (where PERIOD is the name of the ITEM repository member which represents the column) two columns, PERIOD_1 and PERIOD_2, will be generated.

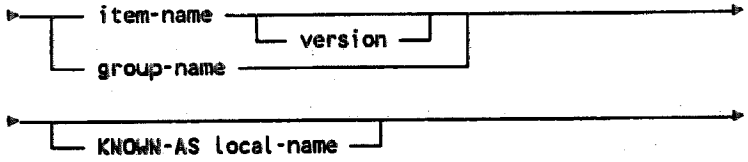
Use the single-column-clause to specify the name of an ITEM or GROUP repository member; each ITEM or GROUP (that is, the ITEMs and/or GROUPs contained by the GROUP) represents a column.

Use the group-name EXPAND clause to specify a GROUP member, which represents one or more columns.

A view, like a table, may have up to 255 generated columns; this figure includes all columns generated as a result of using the "group-name EXPAND" option or the 'column set' option. The clause is checked on generation to ensure that the number of columns specified does not exceed the maximum number permitted.

Use the column-attributes clauses to specify attributes which apply to all columns generated from the preceding clause.

Defining a Single Column in an SQL-VIEW
Use a single-column clause to specify an ITEM or a GROUP repository member which represents a single column. The syntax of the clause is as follows:



item-name is the name of a repository member of the type ITEM; version is the version number of the named ITEM. The version you specify must correspond with a version specified in the ITEM. If no version is specified, then version 1 is assumed by default.

group-name is the name of a repository member of the type GROUP. The GROUP(s) and/or ITEMS(s) contained by this GROUP are combined to represent one column in the view.

The clause is checked on encoding to ensure that the member specified is of one of the correct types (either a GROUP or an ITEM).

Use a KNOWN-AS optional clause to specify a local-name for the ITEM or GROUP. The local-name may be no longer than 18 characters; this is checked on encoding and on generation. This local-name is the name of the column, unless a column-name is specified in the COLUMN-NAME clause.

If no column-name is specified at all in the repository definition (either in the COLUMN-NAME clause, or in the KNOWN-AS clause), then an alias specified in the ITEM or GROUP will be used as the column-name (assuming your environment is tailored to generate aliases as external names). If no alias is specified the ITEM or GROUP member-name will be used as the Name Reduction Process).

Duplicate column names are not permitted. The clause is checked on generation to ensure that no duplicate column-names are present.

EXPAND: Defining Several Columns at Once in an SQL/DS View
Use a group-name EXPAND clause to specify a number of columns at once. The syntax of the clause is as follows:

where **group-name** is the name of a repository member of the type GROUP. The purpose of this clause is to facilitate the generation of several columns in the view at once; it is a shorthand way of referring to a number of GROUPs and ITEMs (that is, all those contained by the specified GROUP), which will each represent one column in the view. Your installation may already have GROUPs defined in its repository which are used in existing applications. These GROUPs may represent records or segments that now need to have counterparts in the SQL/DS environment.

Because of the simpler 'flat-file' structures supported by SQL/DS you need to observe the following points.

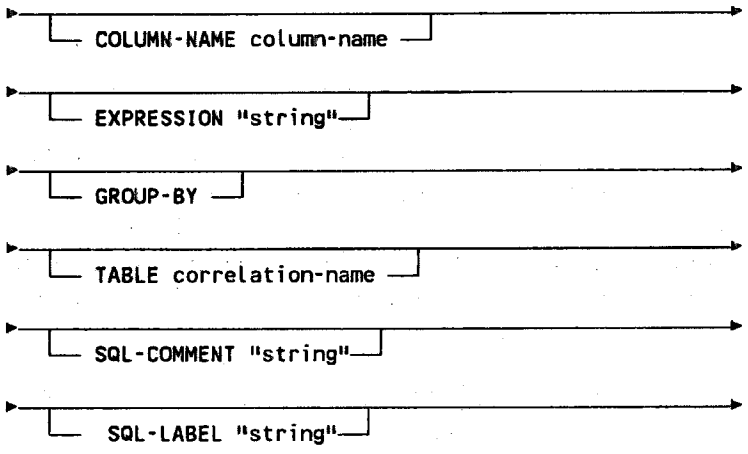
A GROUP to be EXPANDed should not contain any ELSE clauses. These give rise to record 'overlays', that is, records in which certain fields may share the same areas of physical storage. In SQL/DS such a concept has no meaning, since a column in a view must have a name unique in the view and cannot 'overlay' or share data with any other column in the view. If you do have an ELSE clause, it will be ignored.

A GROUP to be EXPANDed may contain 'nested' groups as well as items. Nesting can continue to any depth; the only limit is the amount of memory available. However, whereas in segments and host language data structures, such nesting is meaningful, in an SQL/DS view, it is not. Therefore, intermediate levels in the data structure are removed, in order to generate a 'flat' structure.

You may not specify a KNOWN-AS clause or a version for an EXPANDed group; this is checked on encoding. Therefore, the column-names generated from a group-name EXPAND clause are taken either from the KNOWN-AS names of the ITEMs and/or GROUPs contained by the GROUP specified, or, failing that, from their repository names. The length of any column-name may be no longer than 18 characters; this is checked on encoding. The clause is also checked on generation to ensure that the length of the derived names is compatible with the external environment.

COLUMN-NAME: Defining Column Attributes for an SQL/DS View
A number of optional clauses (the "column-attributes" clauses) are available to specify the attributes of a column or columns defined in a CONTAINS clause.

The column attributes which you specify will apply equally to all of the columns generated from the preceding CONTAINS clause. Therefore, if you have specified a 'column set' or if you have used the 'group-name EXPAND' option, then the column attributes will apply equally to all of the columns so generated. The syntax of these clauses is as follows:



where **column-name** is the name by which the column will be known. The column-name may be no longer than 18 characters; this is checked on encoding. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

If you omit to specify a column-name, then the column-name generated is taken from the KNOWN-AS name, an alias, or the repository member name, of the ITEM or GROUP which represents the column, according to the rules for the derivation of external names.

Note: since each column-name must be unique to a view, you may not use this clause after the 'group-name EXPAND' option.

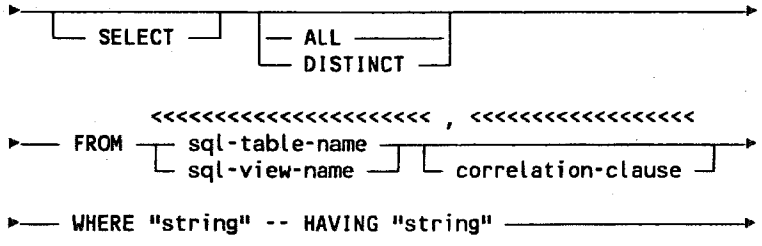
where "string" in the EXPRESSION clause is a valid SQL expression enclosed in quotes. The string can be a maximum of 255 characters long. This string constitutes the expression which is used to calculate the values to be contained in the column (the values being the result of the calculation performed on one or more of the other columns in the view). Where a view contains such a column you should have a corresponding entry in the CONTAINS clause defining an ITEM, from which the correct data type can be derived on generation of an SQL PRODUCE command.

The keyword GROUP-BY is used to indicate that the column is to be a 'GROUP BY' column: this keyword has the same meaning as for SQL/DS.

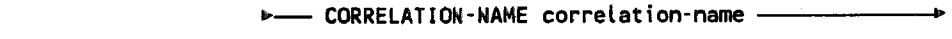
Correlation-name is the correlation-name of the table or view from which a column is being selected. The correlation-name may be no longer than 18 characters; this is checked on encoding. The purpose of this clause is to resolve any ambiguity which may arise when, for instance, two columns with the same name are present in two tables or views specified in the FROM clause. The TABLE clause, therefore, corresponds with the CORRELATION-NAME clause of the FROM clause: the correlation-name given to a table or view in that clause must be the same as the one given to the table or view in this clause.

You can use the SQL-COMMENT and SQL-LABEL clauses in the same way as they are used with regard to a view, except that in this context they apply only to the column(s) defined in the preceding CONTAINS clause.

SELECT: Defining an SQL/DS View Subselect
You can define the subselect of a view in the repository by using the SELECT, FROM, CORRELATION-NAME, WHERE and HAVING clauses. The syntax of the "subselect" clauses is as follows:



correlation-clause is:



In the SELECT clause, the keywords ALL and DISTINCT have the same meanings as in SQL/DS. The clause is checked on encoding to ensure the option specified is a valid one. The keyword SELECT should not be included in the member definition, if neither option is required; if you omit the clause completely, the keywords AS SELECT will be generated with neither ALL nor DISTINCT.

In the FROM clause, sql-table-name is the name of a repository member of the type SQL-TABLE, and sql-view-name is the name of a repository member of the type SQL-VIEW. The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

These are the tables and/or views upon which a view is based. The SELECT clause must be present for the successful generation of an SQL CREATE statement, and the tables and views specified in it must exist, and have valid CREATOR-OWNER clauses.

In the CORRELATION-NAME clause, correlation-name is the correlation-name to be assigned to the sql-table or sql-view specified in the FROM clause. The correlation-name may be no longer than 18 characters; this is checked on encoding. The correlation-name which you define here must correspond with that defined in the TABLE clause. The correlation-name is generated only if the TABLE clause is present.

In the WHERE and HAVING clauses, "string" consists of a valid SQL expression, enclosed in quotes. This string constitutes the 'where-clause' or the 'having-clause' of the subselect in an SQL CREATE VIEW statement. It should contain the valid SQL expression of a 'where' search condition or a 'having' search condition, as appropriate.

Note: the validity of the SQL expression is not checked, when an SQL-VIEW member is encoded, or when an SQL statement is generated.

SQL-COMMENT: Defining an SQL-COMMENT on a View or a Column
Use an SQL-COMMENT clause to specify that a view or a column is to have an associated SQL comment. The syntax of the clause is as follows:

where **"string"** is the comment. The comment must be a character string consisting of no more than 254 characters, enclosed in quotes. The length of the string is checked on encoding.

If you use the SQL-COMMENT clause to qualify a column, it must follow the contains-clause which defines the column. To qualify a view, put the clause at the start of the data definition statement, before the column-specification clauses.

This clause must be present if you want to generate SQL COMMENT ON statements from an SQL-VIEW definition.

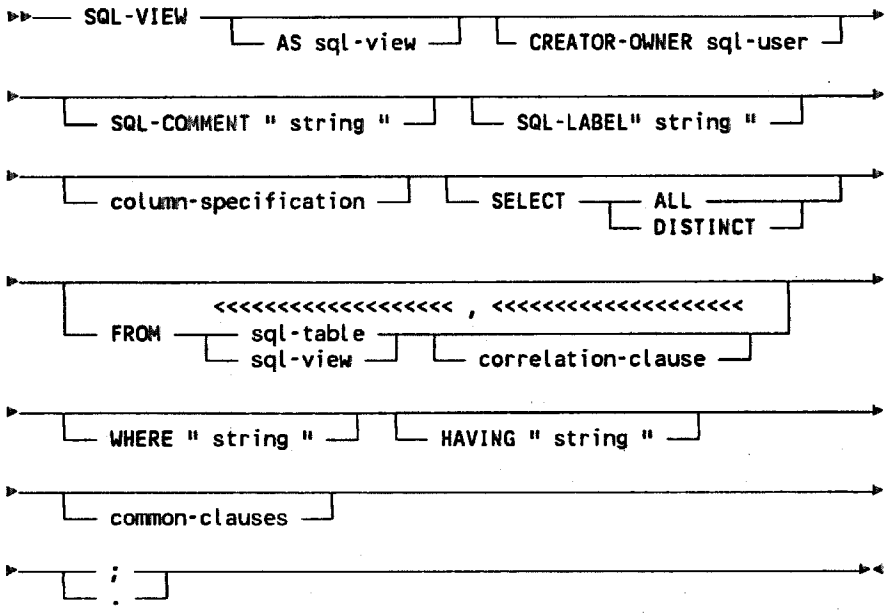
SQL-LABEL: Defining an SQL-LABEL on a View or a Column
Use an SQL-LABEL clause to specify that a view or a column is to have an associated SQL-LABEL. The syntax of the clause is as follows:

where **"string"** is the label. The label must be a character string consisting of no more than 30 characters, enclosed in quotes. The length of the string is checked on encoding.

If you use the SQL-LABEL clause to qualify a column, it must follow the CONTAINS clause which defines the column. To qualify a view, put the clause at the start of the data definition statement, before the column-specification clauses.

This clause must be present if you want to generate SQL LABEL ON statements from an SQL-VIEW definition.

Syntax



where

sql-view is the name of an SQL-VIEW member

sql-user is the name of an SQL-USER member

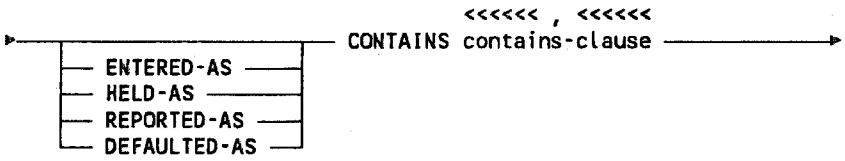
string (in the SQL-COMMENT clause) is a character string of no more than 254 characters

string (in the SQL-LABEL clause) is a character string of no more than 30 characters

string (in the WHERE clause) is a character string

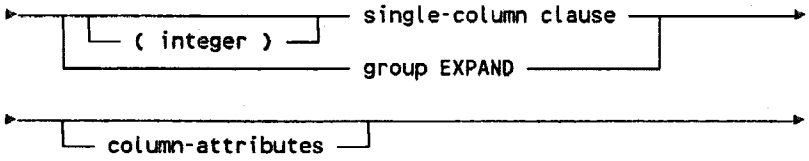
string (in the HAVING clause) is a character string

column-specification is:



where

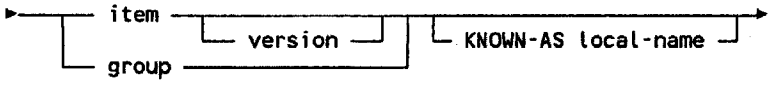
contains-clause is:



where

integer is the number of columns in a 'column set'
group is the name of a GROUP member

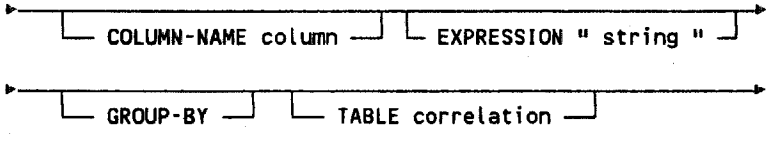
single-column clause is:

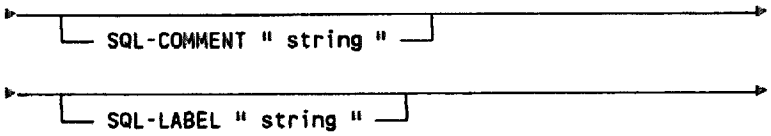


where

item is the name of an ITEM member
group is as defined above
version is an integer in the range 1 to 15
local-name consists of no more than 18 characters

column-attributes is:





where

column is the name of the column, consisting of no more than 18 characters

string (in the EXPRESSION clause) is a character string of no more than 255 characters

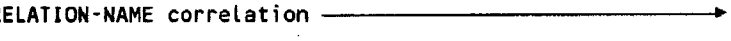
correlation is the name of a correlated table or view, consisting of no more than 18 characters

string (in the SQL-COMMENT clause) is a character string of no more than 254 characters

string (in the SQL-LABEL clause) is a character string of no more than 30 characters

sql-table is the name of an SQL-TABLE member

correlation-clause is:



where

correlation is the name of a correlation name, consisting of no more than 18 characters

common clauses are any of the clauses available to all member types.

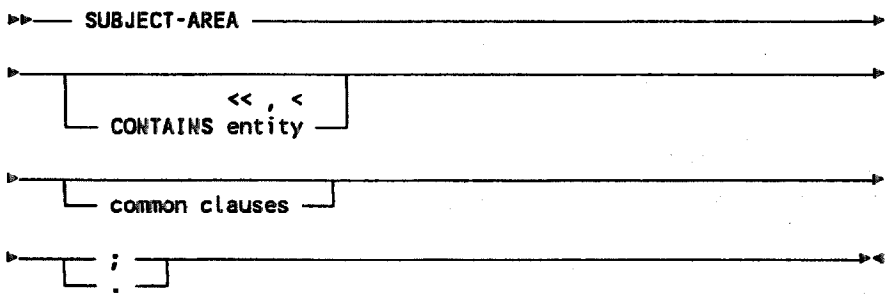
Refer to Appendix 2 for details of the common clauses.

SUBJECT-AREA

A SUBJECT-AREA is a grouping of entities which are logically related because of their involvement in similar business functions.

CONTAINS
Entities named here are said to be logically connected because the data paths between them are used more often than paths to any other entities. Such a grouping of entities constitutes a subject area.

Syntax



where

entity is the name of an ENTITY member

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

SUDS

For details of the SUDS interface member types:

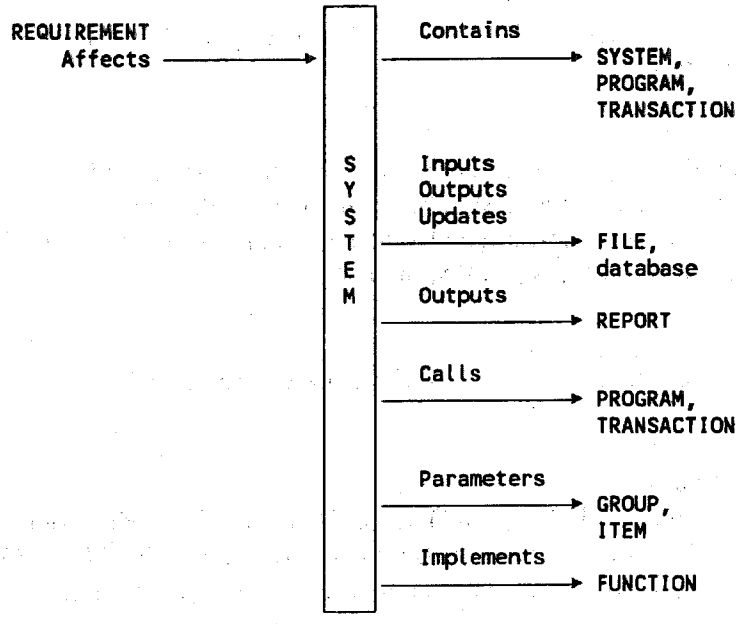
- SUDS-AREA
- SUDS-DATABASE
- SUDS-RECORD
- SUDS-SET
- SUDS-SUBSCHEMA
- SUDS-VIEW

please refer to the Siemens Universal Database System Interface manual (DMR-SUDS).

SYSTEM

A SYSTEM is a set of manual and automated procedures and associated documentation which work together to satisfy one or more informational needs of the organization.

The more commonly used relationships to and from SYSTEM are:



CONTAINS
Contains the name of each lower-level SYSTEM, MODULE or PROGRAM member which composes this higher-level SYSTEM member.

CALLS
This clause may contains the names of PROGRAM and MODULE members.It may also contain one AT subclause, naming the point at which the member is called, and a PASSING clause naming the data members passed to it. Only one AT subclause can be specified in a CALLS clause. To define multiple called members, each with its own AT subclause, multiple CALLS clauses must be specified.

SYSTEM

INPUTS
Contains one or more data-names, of members which input information to this member for processing or reference. It should also list the names of members updated by generation of a new version, rather than replacement of an existing version.

OUTPUTS
Lists the members which receive the information output by the system. It should also list the names of members which this member updates by generating a new version, rather than by replacing an existing version.

UPDATES
The only members named here are those which this system updates by replacing the original version. Members which are updated by generation of a new version should be declared in the INPUTS and OUTPUTS clauses of the SYSTEM member.

DELETES
Contains the names of the members which are deleted by this system.

PARAMETERS
Contains the names of ITEM, GROUP or FILE members which define parameters passed to this member. It may also include an ENTRY-POINT subclause for each data-name, containing the label of an entry point in this member.

IMPLEMENTS
Contains the name of each function implemented by the system.

CATEGORY
Indicates the business level to which reports from the system apply; for example, operational level (payroll), design support (forecasting), or top strategic level (product decisions).

COMMITTED-ENHANCEMENTS
Records approved or in-progress enhancements to a system. This information is needed at Strategic Information Planning (SIP) level, to ascertain existing and required systems.

DISPOSITION
Contains a string value up to 78 characters long and describes the status of the SYSTEM member. There is no default value for this clause, but suggested values are EXISTING, PLANNED and NEWLY-PLANNED.

INVESTMENT
The value held is an estimate (in dollars) of the cost of this system (new or planned) when finished.

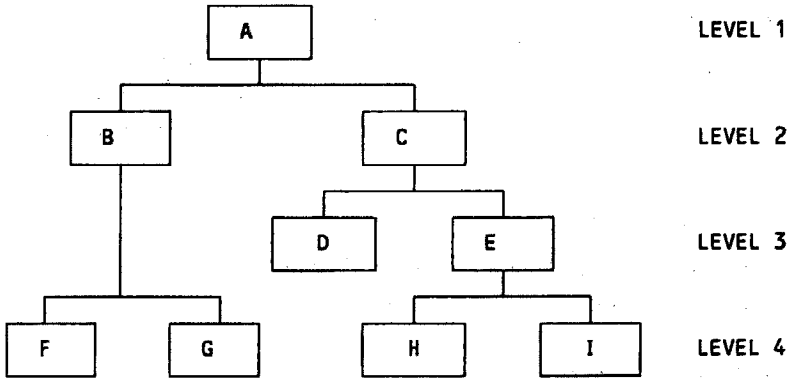
NET-PRESENT-VALUE
The value held is an estimate (in local currency) of the value of this system to the enterprise. It could be used when calculating the benefit of a system, or the value of enhancing it.

OPERATING-MODE
The clause contains the name of the operating mode used for the system.

RETURN-ON-INVESTMENT
Holds an estimate (in local currency) of the return which can be expected from investing in this system. The value could be used when calculating the benefit of a system.

WEAKNESSES
Documents anticipated or known weaknesses of the system which may need to be taken into account if risk analysis is performed, or when applications are being planned.

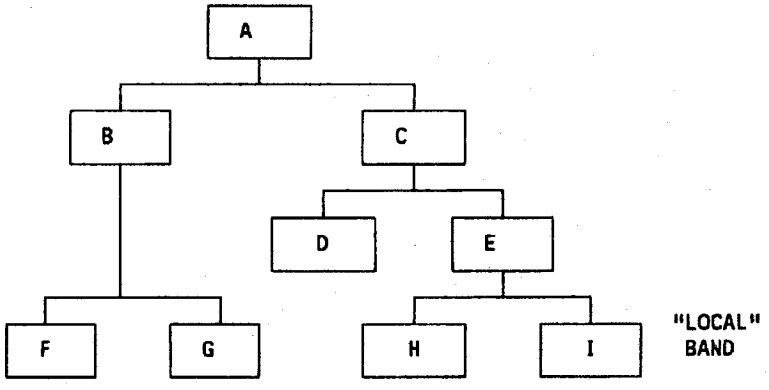
LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy.
The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

BAND

The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.

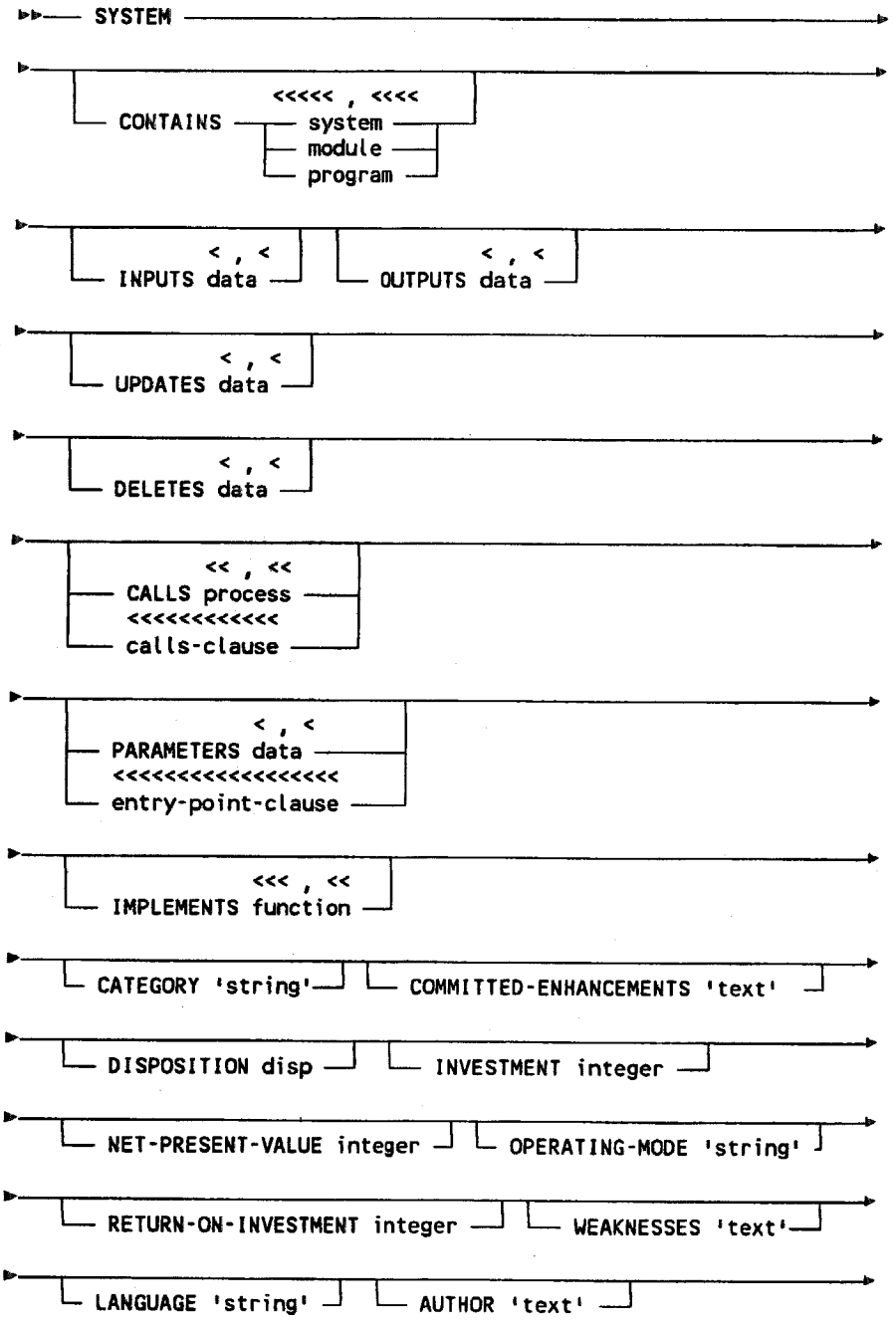


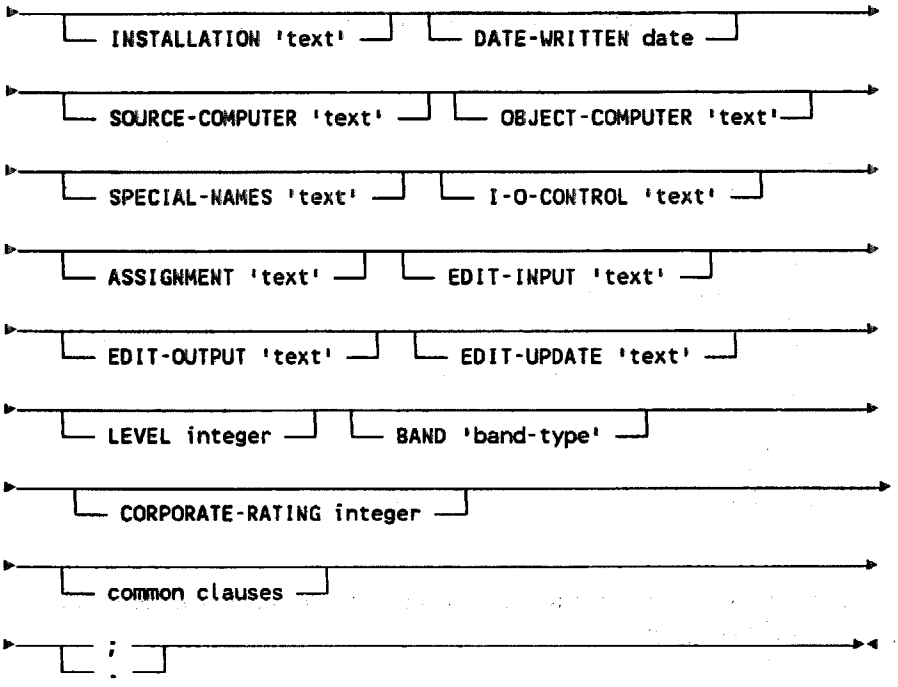
The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

CORPORATE-RATING
Stores an integer reflecting the importance that the enterprise attaches to this member. Any scale of values can be used, but it must be used for every system or proposed system in the enterprise.

SYSTEM

Syntax

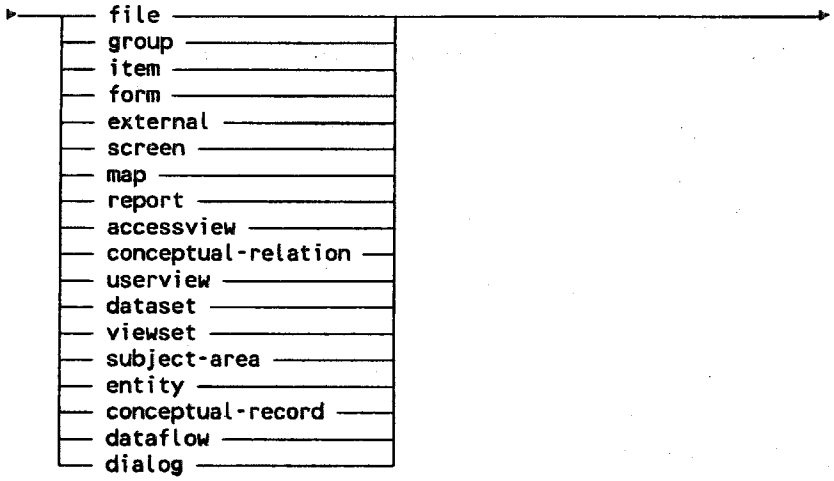




where

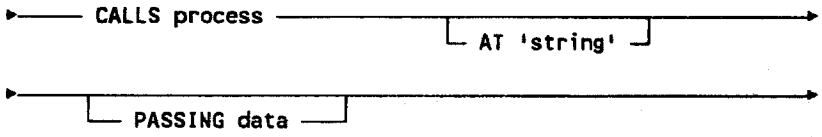
system is the name of a SYSTEM member
module is the name of a MODULE member
program is the name of a PROGRAM member

data is the name of a member of any of the following types:



process identifies a process member at the same or a lower level in the member type hierarchy

calls-clause is:



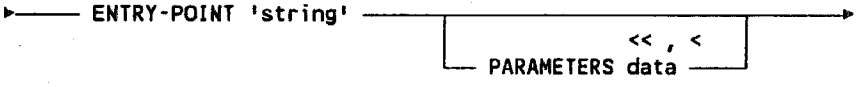
where

process is as defined above

'string' is a character string of not more than 256 characters

data identifies a data member

entry-point-clause is:



where **'string'** and data are as defined above

function is the name of a FUNCTION member

'string' is a character string of not more than 256 characters

'text' is a maximum of 32767 delimited character strings, each containing a maximum of 246 characters

disp is a string containing a maximum of 78 characters

integer is an integer value of up to 18 digits, optionally preceded by a sign

date is a date, in the format defined by your installation

'band-type' is a text string of up to 78 characters, including delimiters.

Note: the delimiters shown in the above syntax are required when defining a SYSTEM member via the command interface.

integer is an integer value of up to 18 digits, optionally preceded by a sign

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

SYSTEM-2000

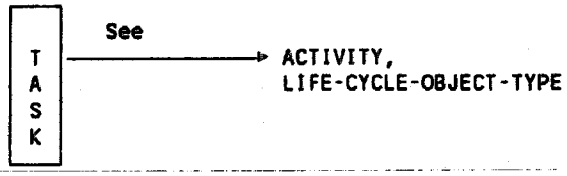
For details of the System 2000\80 interface member types:

SYSTEM-2000-DATABASE
SYSTEM-2000-SCHEMA-RECORD
SYSTEM-2000-SUBSCHEMA-RECORD

please refer to the System 2000\80 Interface manual (DMR-S2K80)

TASK

A TASK is a piece of work which has a definite beginning and end and results in one or more identifiable products; the basic building block of work in a project.
The more commonly used relationships to and from TASK are:



OPTION
Short description of project management components, as shown on the user interface.

OPTION-TEXT
Long description of project management components, as shown on the user interface.

EXECUTANT
Name of the employee who is responsible for the execution of the task.

PLANNED-BEGIN
Estimated date at which the task will be started.

ACTUAL-BEGIN
Actual date at which the task started.

PLANNED-END
Estimated date at which the task will be finished.

ACTUAL-END
Actual date at which the task is finished.

ESTIMATED-DURATION
Determined or estimated value of how long the task execution takes.

TASK

ACTUAL-DURATION
Actual value of how long the task execution has been taken.

INPUTS
Contains the names of any ENTITY and ACCESSVIEW members which input information to this member for processing or reference. It should also list the names of members updated by generation of a new version, rather than replacement of an existing version.

OUTPUTS
Lists the names of members to which this function writes information used for processing or reference. It should also list the names of members which are updated by generating a new version, rather than replacing an existing version.

UPDATES
The only ENTITY members named here are those which this function updates by replacing the original version. Members which are updated by generation of a new version should be declared in the INPUTS and OUTPUTS clauses of the function.

DELETES
This clause may contain the names of one or more ENTITY members, which are deleted by this function.

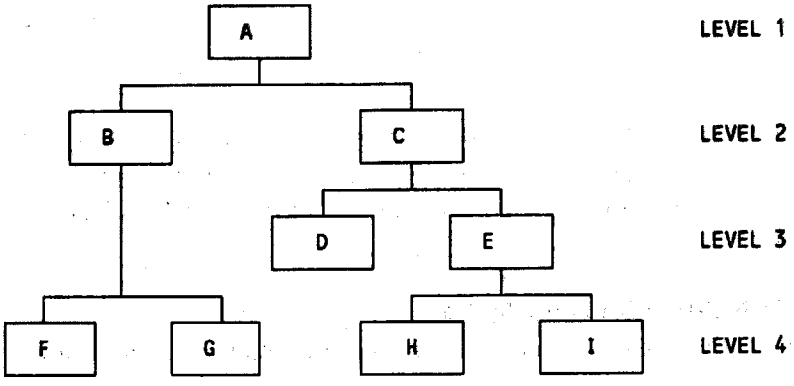
CALLS
The PROCESSING-RULE, PROGRAM and MODULE members named in this clause are called when the function is performed. This information can be used to find out which processes are common to different functions.

CONTAINS
Must contain the name of every function which is a part of this function.

SUPPORTS
Contains the names of one or more critical success factors.

LOCATION
States the location within the enterprise of this function, by naming the appropriate LOCATION members.

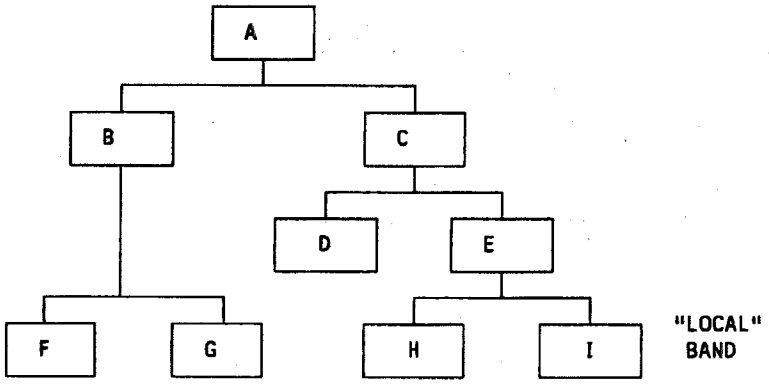
LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy.
The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

BAND
The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.

TASK



The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

POSSIBLE-OPERATING-MODE
Contains the name of the possible future operating mode used for the function.

RESPONSIBLE-PERSON
Identifies the person responsible for the performance of the function. It is recommended that the person's job title is used rather than his or her name.

RESPONSIBILITY-DESCRIPTION
Describes the duties and responsibilities of the person named in the RESPONSIBLE-PERSON clause of this FUNCTION member.

HELP
In this clause you can give a more specific description of the member, its function etc.
Note: You may enter one or more lines of text without using single quotes (').

Syntax

The syntax of the TASK member type will be supplied in a future issue.

TOTAL

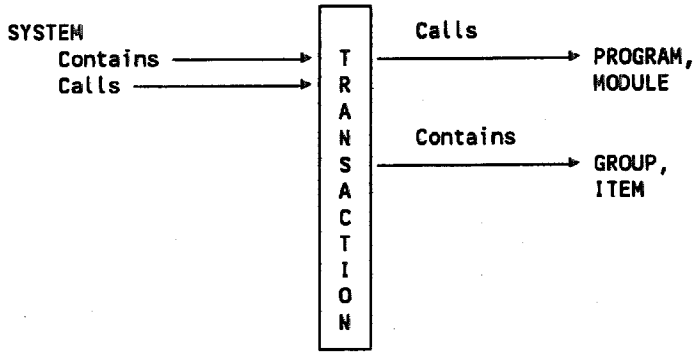
For details of the TOTAL interface member types:

TOTAL-DATABASE
TOTAL-FILE
TOTAL-MASTER-FILE
TOTAL-VARIABLE-FILE

please refer to the TOTAL Interface manual (DMR-TOT).

TRANSACTION

A TRANSACTION is a specific set of input data that triggers the execution of a program.
The more commonly used relationships to and from TRANSACTION are:



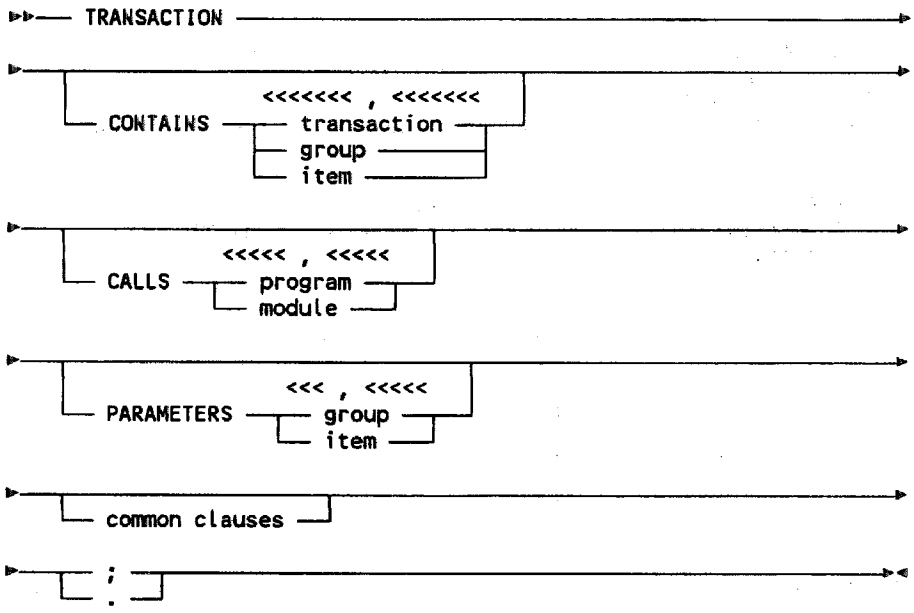
CONTAINS
Contains the name of each TRANSACTION, GROUP or ITEM member which initiates the program or module named in the CALLS clause of this member.

CALLS
The PROGRAM and MODULE members which may be named in this clause are called as a result of an event in this transaction.

PARAMETERS
Contains the names of ITEM, GROUP or FILE members which define parameters passed to this member. It may also include an ENTRY-POINT subclause for each data-name, containing the label of an entry point in this member.

TRANSACTION

Syntax



where

transaction is the name of a TRANSACTION member

group is the name of a GROUP member

item is the name of an ITEM member

program is the name of a PROGRAM member

module is the name of a MODULE member

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

USERVIEW

Use the USERVIEW member type to define the access paths between data elements required by a user in a particular application-oriented view of data.

A USERVIEW member of the modeling repository is used to define the relationships that exist amongst the data elements required in a particular application or in a user's view of that application. It is formulated as a set of dependencies (functional, multivalued and/or domain dependencies - FDs, MVDs and DDs) that hold between sets of (one or more) data elements. FDs and MVDs are defined in DEPENDENCIES attributes and DDs in DOMAIN attributes.

The relationships expressed by the dependencies can include both access paths and logical relationships that hold between the data elements. This can also indicate (in the case of a DD) that one data element is a subcategory of another.

In addition to representing a list of dependencies, a USERVIEW member can be used to specify estimates of the relative frequency of access for the USERVIEW's data elements and the required response time. Also, for each MVD in the USERVIEW, an estimate can be specified of its multiplicity, that is, the average number of values determined for its right-hand side by a given value of its left-hand side.

Load factor calculations can be made from these estimated performance parameters, if you have the Load Factors Function.

Refer to the Load Factor Calculation manual for details of the Load Factors Function.

As in the case of any repository member type, common clauses may also be specified. Not more than one of each of these clauses can be declared. They can be declared in any order and in any position between the statement identifier and the terminator, except that they must not appear within DEPENDENCIES, DOMAIN or SUBDOMAIN attribute. Nor may they appear between any associated DOMAIN and SUBDOMAIN attributes.

Refer to Appendix 2 for details of common clauses.

DEPENDENCIES: Specifying Functional and Multivalued Dependencies Between Data Elements

Use each DEPENDENCIES attribute to represent a list of one or more functional dependencies (FDs) and/or multivalued dependencies (MVDs), all having the same left-hand side (LHS) and differing right-hand side (RHS). Each RHS attribute in a DEPENDENCIES attribute declares a listed set of one or more data elements as being functionally or multiply-dependent on the set of data elements named in the LHS attribute. Consecutive names in a LHS or RHS attribute are separated by commas.

For example, the following attribute:

```
DEPENDENCIES
LHS EMP-NO  FD  RHS DEPT-NO
            MVD RHS CHILD-NAME
            FD  RHS EMP-ADDRESS
            MVD RHS PAST-SAL, SAL-DATE
```

would result in placement of the following dependencies in the workbench design area by a subsequent merge.

```
EMP-NO ——→ DEPT-NO
EMP-NO ——→→ CHILD-NAME
EMP-NO ——→ EMP-ADDRESS
EMP-NO ——→→ PAST-SAL + SAL-DATE
```

All data element names appearing in a DEPENDENCIES attribute must conform to the rules for coding repository member names. In particular, users who have the Top-down Modeling Function installed should observe the indicated restraints with respect to naming data elements.

Refer to the Enterprise Modeling manual for details of the Top-down Modeling Function.

Refer to the CONTROLMANAGER User's Guide (CMR-UG) for the naming rules.

DOMAIN: Specifying Domain Dependencies Between Data Elements
Each DOMAIN attribute followed by one or more SUBDOMAIN attributes represents a list of one or more DDs all having the same right-hand side. That is, each SUBDOMAIN attribute specifies a listed set of one or more data elements as a subcategory of the set of data elements named in the preceding DOMAIN attribute. Consecutive data elements in either attribute are separated by commas.

As an example, the following specification:

```
DOMAIN IS EMP-NO
       SUBDOMAIN IS MGR-EMP-NO
       SUBDOMAIN IS ADMIN-EMP-NO
```

would declare MGR-EMP-NO and ADMIN-EMP-NO as subcategories of EMP-NO and would result in the following DDs being placed in the workbench design area by action of a subsequent merge.

```
MGR-EMP-NO ——→ EMP-NO
ADMIN-EMP-NO ——→ EMP-NO
```

It is recommended for paired DOMAIN and SUB-DOMAIN attributes to each contain the same number of data elements.

All data element names appearing in a DOMAIN or SUBDOMAIN attribute must conform to the rules for coding repository member names. In particular, users who have the Top-down Modeling Function installed should observe the indicated constraints with respect to naming ENTITIES and data elements.

RELATIVE-FREQUENCY, RESPONSE-TIME: Specifying Performance Parameters for Use in Calculating Load Factors
Use the optional RELATIVE-FREQUENCY and RESPONSE-TIME attributes to specify, respectively, your estimates of:

- the relative frequency of access to the data elements named in a USERVIEW member definition

- the response time required for data to be accessed by the process represented by the USERVIEW.

If RELATIVE-FREQUENCY and/or RESPONSE-TIME attributes appear in the USERVIEW definition, they must precede all DEPENDENCIES, DOMAIN and SUBDOMAIN attributes.

Similarly, for each MVD defined (in a DEPENDENCIES attribute of the USERVIEW), an estimate can be specified of its multiplicity, the average number of values of its right-hand side determined by a given value of its left-hand side.

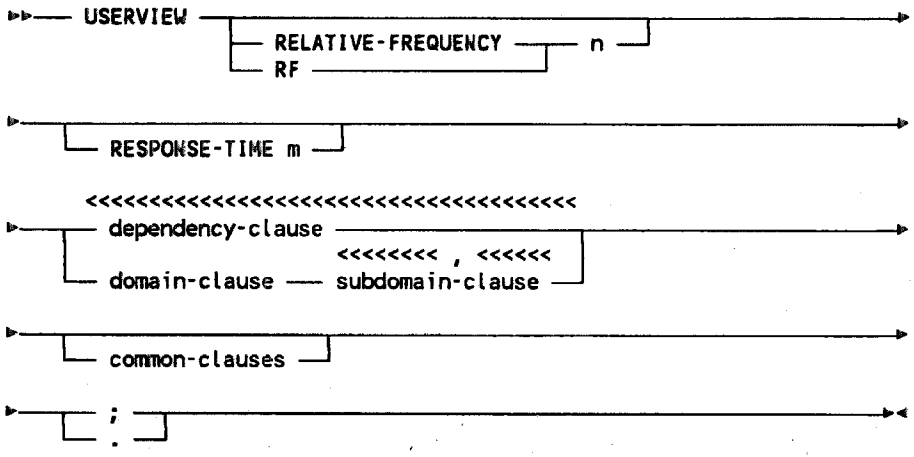
These performance parameters can be used to calculate the following load factors for each data element appearing in one or more of the USERVIEWS present in the workbench design area (WBDA):

- total access frequency, accumulated over all the USERVIEWS in the WBDA

- weighted average response time, averaged over all the USERVIEWS.

The contribution from a USERVIEW with an unspecified performance parameter is taken to be zero for that parameter.

Syntax

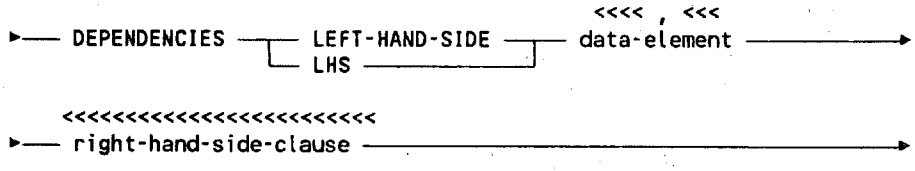


where

n is an unsigned integer indicating the relative frequency of access to the USERVIEW being defined

m is an unsigned integer indicating the response time required for data accessed by the process represented by the USERVIEW

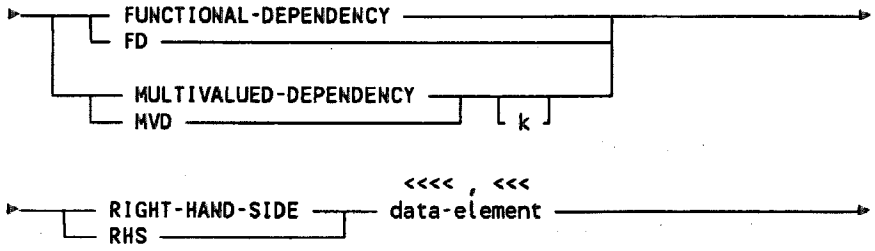
dependency-clause is:



where

data-element is the name of a data element that is held in the modeling repository as an ITEM or GROUP member or is to be entered in the modeling repository as a dummy ITEM

right-hand-side-clause is:

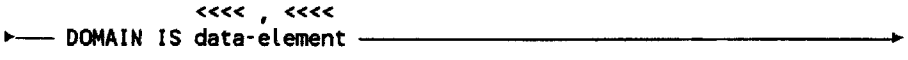


where

k is an unsigned integer indicating the multiplicity for a multivalued dependency, that is, the average number of values (or sets of values) of the right-hand side of the dependency determined by a given value of the left-hand side. If not specified for a dependency, the default value of k is taken to be 1. (The multiplicity of a functional dependency is automatically taken to be 1.)

data-element is as defined above

domain-clause is:



subdomain-clause is:

where **data-element** is as defined above

common clauses are any of the clauses common to all member types

Refer to Appendix 2 for details of the common clauses.

VIEWSET

The VIEWSET member type can be used to represent a collection of data-views grouped into logically related application areas or subject areas of an enterprise.

A VIEWSET member of the modeling repository represents a collection of data-views defined by the USERVIEW, ENTITY and/or subordinate VIEWSET members named in the VIEWSET CONTAINS attribute.

The primary purpose of a VIEWSET member is to group data-views into logically related sets, for example, all USERVIEWS that describe the operation of a personnel department. You can group the data-views into subject areas of the enterprise being modeled or into application areas of the database being designed.

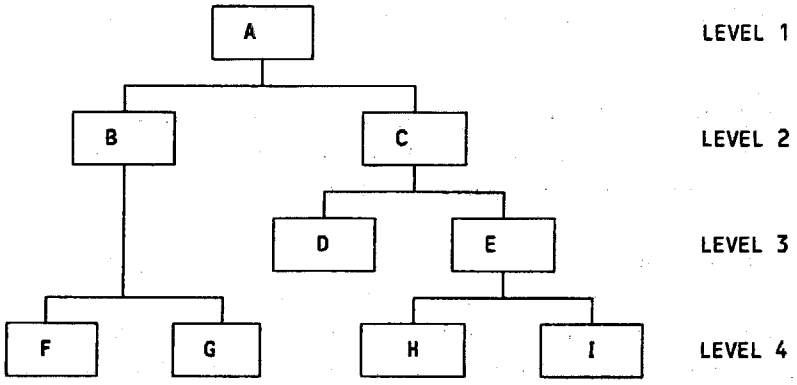
A VIEWSET can also provide a convenient way of merging a large number of data-views from the modeling repository into the workbench design area. That is, instead of naming each USERVIEW and ENTITY member separately for merging, the same result can be obtained by specifying a single VIEWSET to be merged.

CONTAINS
The optional CONTAINS attribute is used to list the names of USERVIEW, ENTITY and/or subordinate VIEWSET members that define, both directly and indirectly, the data-views being modeled. (VIEWSETS can be nested in this manner to any depth.) Consecutive member names in the list must be separated by commas and optionally by spaces.

All names specified in the **CONTAINS** attribute must conform to the rules for naming repository member names. In particular, users who have the Top-down Modeling Function installed should observe the indicated constraints with respect to naming ENTITIES and data elements.

LEVEL
Contains the integer counterpart of the BAND clause, which is assigned to members for selection and grouping. It could represent the level a member occupies in a hierarchy, or be used to group members from different physical tiers in a member hierarchy.

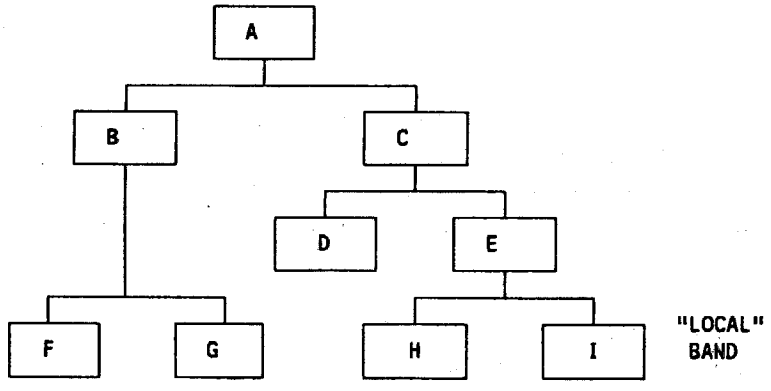
The diagram and notes below illustrate how the LEVEL attribute can be used to logically link members from different tiers in a hierarchy.



Members F,G,H and I have a LEVEL clause with a value 4. They can be grouped together via this clause, even though H and I are at a different tier of decomposition from F and G. There is no default value for the LEVEL clause.

BAND

The clause takes a string value which can group members for processing (such as LOCAL or NEW), irrespective of which tiers they occupy in a hierarchy. The diagram and notes below illustrate how BAND can be used to logically link members from different tiers in a hierarchy.

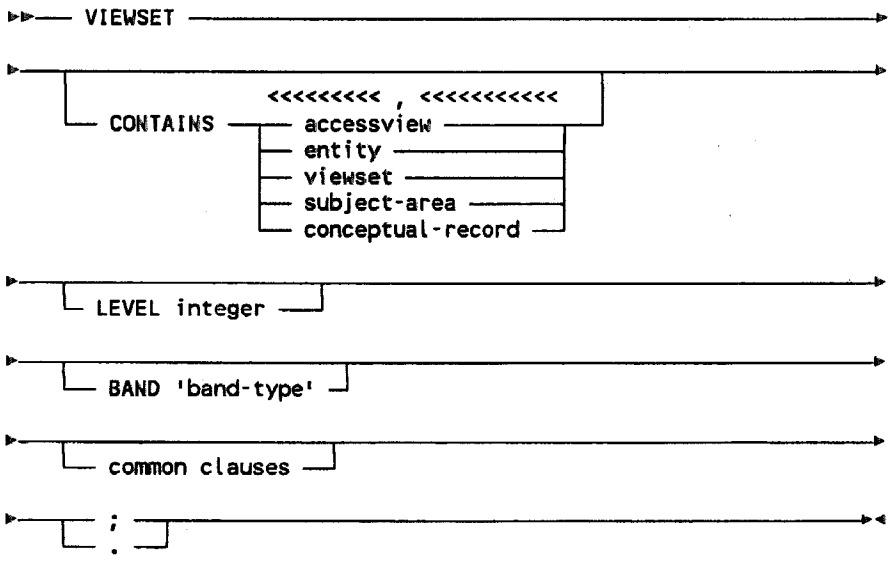


The members F, G, H and I have a BAND clause containing "LOCAL". They can be grouped together via this clause, despite different tiers of decomposition. There is no default value for the BAND clause.

As in the case of any repository member type, common clauses may also be specified. Not more than one of each of these clauses can be declared. They can be declared in any order and in any position between the statement identifier and the terminator, except that they must not appear within a CONTAINS attribute.

VIEWSET

Syntax



where

accessview is the name of an ACCESSVIEW member

entity is the name of an ENTITY member

viewset is the name of a VIEWSET member

subject-area is the name of a SUBJECT-AREA member

conceptual-record is the name of a CONCEPTUAL-RECORD member

integer is an integer value of up to 18 digits, optionally preceded by a sign.

'band-type' is a text string of up to 78 characters, including delimiters.

common clauses are any of the clauses common to all member types.

Refer to Appendix 2 for details of the common clauses.

APP. 1 ENTERPRISE ENTITY-RELATIONSHIP DATA SUBMODEL

APP.1.1 INTRODUCTION

You use the Enterprise Entity-Relationship Data Submodel to create an Entity-Relationship (ER) model of data requirements of an enterprise or an aspect of an enterprise. Such a model is here termed a business model.

A business model is represented in the repository using the following member types:

```
ENTITY
BUSINESS-RELATIONSHIP
HAS-ATTRIBUTES
GROUP
ITEM.
```

Section App.1.2 describes business models.

Section App.1.3 introduces the member types that you use to represent business models in the repository.

Section App.1.4 describes typical constructs in business models and how to represent them in the repository.

APP.1.2 BUSINESS MODELS

App.1.2.1 Introduction

At the highest level, the model consists of:

- business entities
- business relationships
- attributes
- has-attributes relationships.

In describing business models, the names of business entities, business relationships and attributes are given in quotes. In describing the representation of models in the repository member names and member types are, as always, given in block capitals.

Business entities specify the people, places, things, and concepts that are important to the business. For brevity, after this section, "entity" is used for "business entity" wherever there is no ambiguity.

Business relationships specify significant relationships between business entities or business relationships. In their simplest form they merely record the existence of a relationship. For example, a business relationship "supplies" can relate a business entity "supplier" to a business entity "product". That is, "supplier" supplies "product".

A business relationship can have properties giving additional requirements on the two business entities it connects. For example, using the mandatory property a business relationship "supplies" can specify that a business entity "supplier" always exists in association with another business entity "product". That is, "supplier" supplies "product", and a product's supplier must be recorded.

The properties of business relationships are described in section App.1.2.2.

An **attribute** represents a real-world characteristic of a business entity or business relationship. For example, a business entity "payment" might have an attribute "payment-amount".

Each attribute takes one of a range of possible values. For example, you can define an attribute "date" with a range of values from 1st January 1900 to 31st December 1999.

A business entity can have an **identifying attribute**, a value assignment of which uniquely identifies an instance of the business entity. For example, a business entity "payment" might have an identifying attribute "payment-code". An example of an identifying attribute is given in section App.1.4.3.

For further details of identifying attributes refer to the Enterprise Modeling manual (DSR-ENT).

Has-attributes relationships associate an attribute with a business entity or business relationship. These are sometimes referred to as attributive or characteristic relationships. Has-attributes relationships, like business relationships, can have properties. These properties are described in section App.1.2.3.

Figure App.1.1 shows an example model of an enterprise. For clarity the following are omitted from the diagram:

- inverse names
- attributes.

The main points represented by the model are as follows:

- suppliers supply raw materials to the company. A raw material can be supplied by more than one supplier. A supplier can supply more than one raw material. The pricing of the raw material may not be uniform. For example, the first 100 tons may be $50 a ton, the second 100 tons $45 a ton, and so on.

- a supplier invoices the company for an order and sends shipment records as instalments of the order are dispatched

- the company pays the supplier by instalment, not necessarily as instalments of orders arrive. The company keeps its own record of accounts with each supplier.

This example is used in section App.1.4, which shows how various typical constructs are represented in the repository. The complete representation of the model is included in the SAMPLE repository.

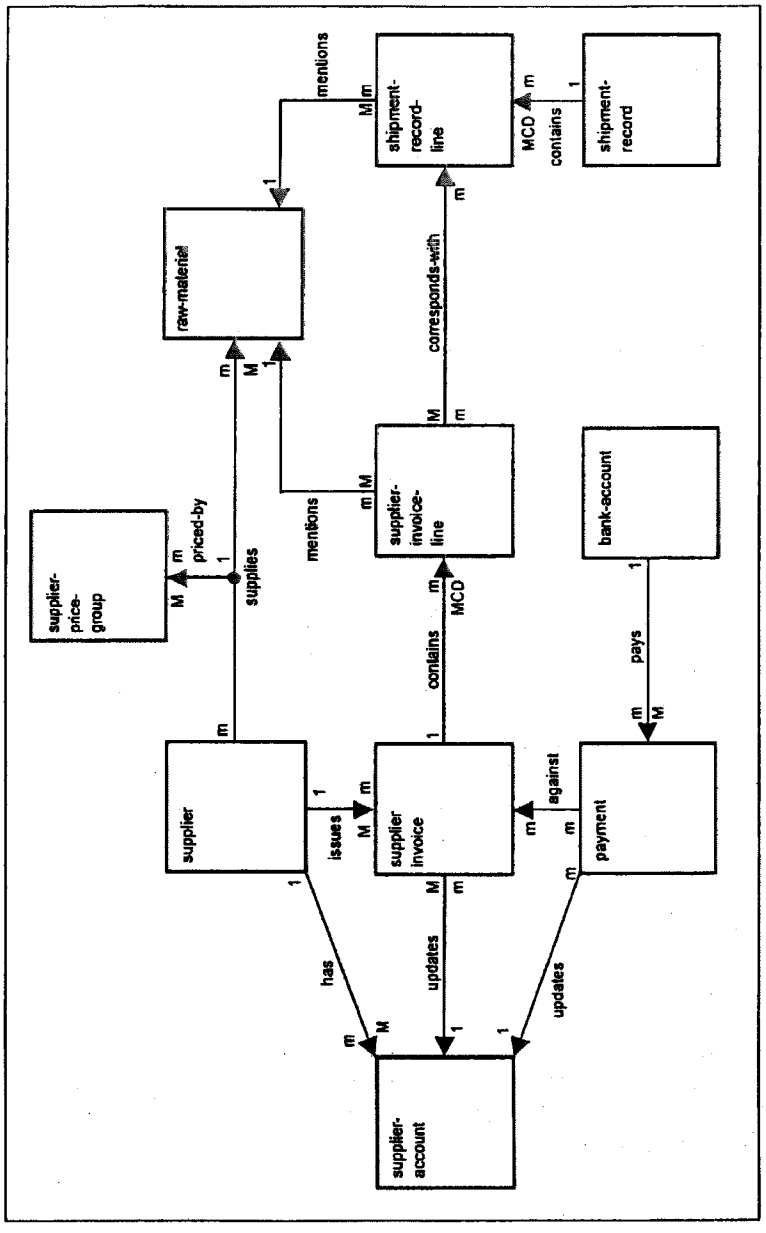


Figure App.1.1 Model of an Enterprise

App.1.2.2 Properties of Business Relationships

A business relationship can have the following properties:

- mandatory
- cardinality
- dependent
- controlled.

These properties are described below. Examples of the properties are given in section App.1.4.

Suppose a business relationship "r" connects one entity "a" (the source) to another entity "b" (the target).

If the source of "r" is **mandatory** then an instance of "a" must be the source of an instance of "r".

Similarly, if the target of "r" is mandatory then an instance of "b" must be the target of an instance of "r".

If the source of "r" has a **cardinality of p,q** then an instance of "b" must be the target of zero or between p and q instances of "r" with instances of "a".

Similarly, if the target of "r" has a cardinality of p,q then an instance of "a" must be the source of zero or between p and q instances of "r" with instances of "b".

Source or target cardinality of p is short for source or target cardinality of 1,p. That is, if a minimum cardinality is 1, it is for brevity omitted.

If the source of "r" is **dependent** then "a" is dependent on "b", that is, "a" is identified by "b"'s identifier and its own identifier.

Similarly, if the target of "r" is dependent then "b" is dependent on "a", that is "b" is identified by "a"'s identifier and its own identifier.

If the source of "r" is controlled then when the last valid instance of "r" is deleted the participant instance of "a" is deleted too. The last valid instance depends on the minimum source cardinality of "r", as follows: if the minimum source cardinality of "r" is n (an integer) then the instance of "r" is the last valid instance if n-1 other instances of "r" have "a" as their source.

Similarly, if the target of "r" is controlled then when the last valid instance of "r" is deleted the participant instance of "a" is deleted too. The last valid instance depends on the minimum target cardinality of "r", as follows: if the minimum target cardinality of "r" is n (an integer) then the instance of "r" is the last valid instance if n-1 other instances of "r" have "a" as their target.

App.1.2.3 Properties of Has-Attributes Relationships

A has-attributes relationship can have the following properties:

- mandatory
- cardinality.

These properties are described below. Examples of the properties are given in section App.1.4.

Suppose a has-attributes relationship "r" connects an entity "a" (the source) to an attribute "b" (the target).

If "r" is **mandatory** then "b" is a mandatory attribute of "a", that is for every instance of "a", "b" must be assigned a value.

If "r" has a **cardinality of p,q** then for every instance of "a", "b" must be assigned a value between p and q times. It can equally be said that "b" has a cardinality of p,q.

Cardinality of p is short for cardinality of 1,p. That is, if a minimum cardinality is 1, it is for brevity omitted.

APP.1.3 THE MEMBER TYPES

A business model is represented in the repository using the following member types:

```
ENTITY
BUSINESS-RELATIONSHIP
HAS-ATTRIBUTES
GROUP
ITEM.
```

An ENTITY member represents an entity.

A BUSINESS-RELATIONSHIP member represents a business relationship.

An ITEM or GROUP member represents an attribute.

A HAS-ATTRIBUTES member represents a has-attributes relationship.

These member types are described in Chapter 2.

Figure App.1.2 shows the schema diagram of the submodel. It shows the allowed relationships via the BUSINESS-RELATIONSHIP and HAS-ATTRIBUTES relationship types. For example, a BUSINESS-RELATIONSHIP member can have an ENTITY member or another BUSINESS-RELATIONSHIP member as its source or its target.

Note that concepts like identifying attribute or subentity can also be represented. For details of these additional concepts refer to the Enterprise Modeling manual (DSR-ENT).

APP.1.4 TYPICAL BUSINESS CONSTRUCTS

App.1.4.1 Introduction

Section App.1.4 describes some typical constructs in business models and how to represent them in the repository. The following is given for each construct:

- the business diagram
- the repository meta-data diagram
- the repository member definitions.

The diagrams and member definitions given are extracts from the example in Figure App.1.1. Details from the example that are not relevant to the point being discussed are omitted wherever possible. So for example the definitions of entity members are usually skeleton definitions.

The following abbreviations are used in business diagrams:

- O for optional
- M for mandatory
- I for independent
- D for dependent
- U for uncontrolled
- C for controlled.

Note the following about business diagrams:

- cardinality is always given in full
- O for optional can be omitted
- I for independent can be omitted
- U for uncontrolled can be omitted.

The defaults for the mandatory, dependent and controlled properties are the same as the equivalent defaults in the BUSINESS-RELATIONSHIP and HAS-ATTRIBUTES member types.

Note the following about meta-data diagrams:

- for entity members the member type and member name are given
- for BUSINESS-RELATIONSHIP members the member type and forward verb are given
- for HAS-ATTRIBUTES members only the member type is given
- important clauses and their values are given beneath the entity or relationship.

The names of BUSINESS-RELATIONSHIP and HAS-ATTRIBUTES members have no semantic significance and are not shown in diagrams. However, they must of course be valid member names.

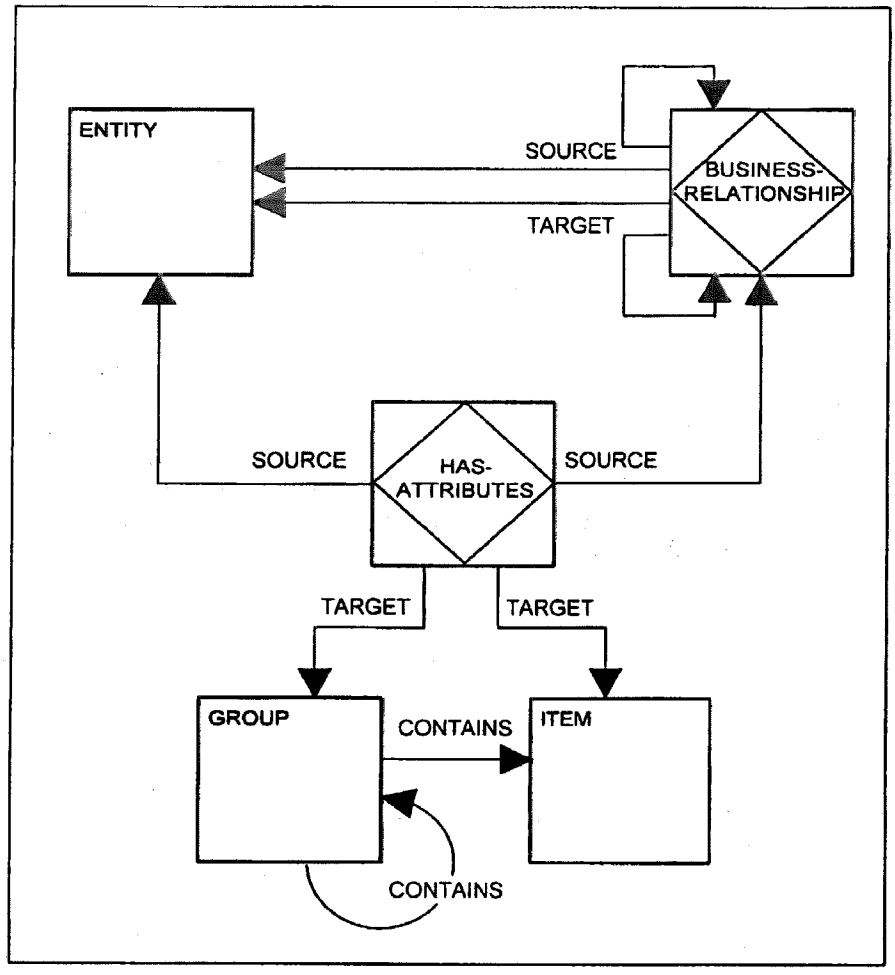


Figure App.1.2 Diagram of the RIM Submodel

App.1.4.2 Unattributed Entity

An unattributed entity "payment" is shown in Figure App.1.3. This is represented in the repository as shown in Figure App.1.4 by an ENTITY member named EN-PAYMENT. The equivalent member definition is given below:

```
REPLACE EN-PAYMENT.
ENTITY
.
```

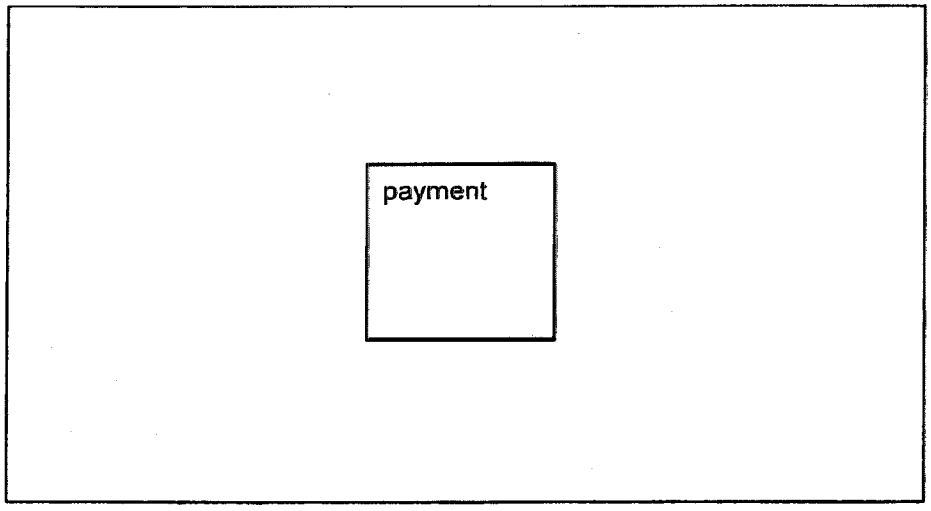


Figure App.1.3 Unattributed Entity

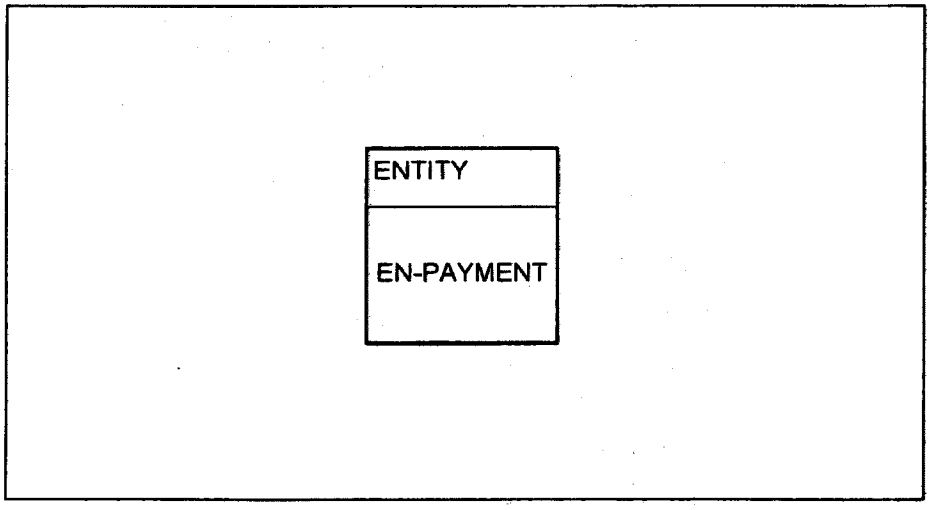


Figure App.1.4 Representation of Unattributed Entity

App.1.4.3 Attributed Entity

An attributed entity is shown in Figure App.1.5. The entity "payment" has two attributes "payment-amount" and "payment-check-number". There is also an identifying attribute "payment-code", which is not shown in the diagram.

Note the following:

- the attribute "payment-amount" is mandatory, that is, for every instance of "payment", "payment-amount" must be assigned a value

- the attribute "payment-check-number" is optional, since a payment need not be made by cheque

- the cardinality of "payment-amount" is 1,1, that is, for every instance of "payment", "payment-amount" must be assigned precisely one value.

This is represented in the repository as shown in Figure App.1.6. The two HAS-ATTRIBUTES members associate the ENTITY member with the two ITEM members. The equivalent member definitions are given below:

```
REPLACE EN-PAYMENT.
ENTITY
IDENTIFIER IS IT-PAYMENT-CODE
.
REPLACE X001.
HAS-ATTRIBUTES
SOURCE EN-PAYMENT
TARGET IT-PAYMENT-CHEQUE-NUMBER
ATTRIBUTE-MANDATORY O
MINIMUM-CARDINALITY 1
MAXIMUM-CARDINALITY '1'
.
REPLACE X002.
HAS-ATTRIBUTES
SOURCE EN-PAYMENT
TARGET IT-PAYMENT-AMOUNT
ATTRIBUTE-MANDATORY M
MINIMUM-CARDINALITY 1
MAXIMUM-CARDINALITY '1'
.
REPLACE IT-PAYMENT-CODE.
ITEM
.
```

(continued)

(continued)

```
REPLACE IT-PAYMENT-CHEQUE-NUMBER.
ITEM
.
REPLACE IT-PAYMENT-AMOUNT.
ITEM
.
```

Note that clauses specifying a maximum cardinality are text clauses so that the non-specific m for many can be used.

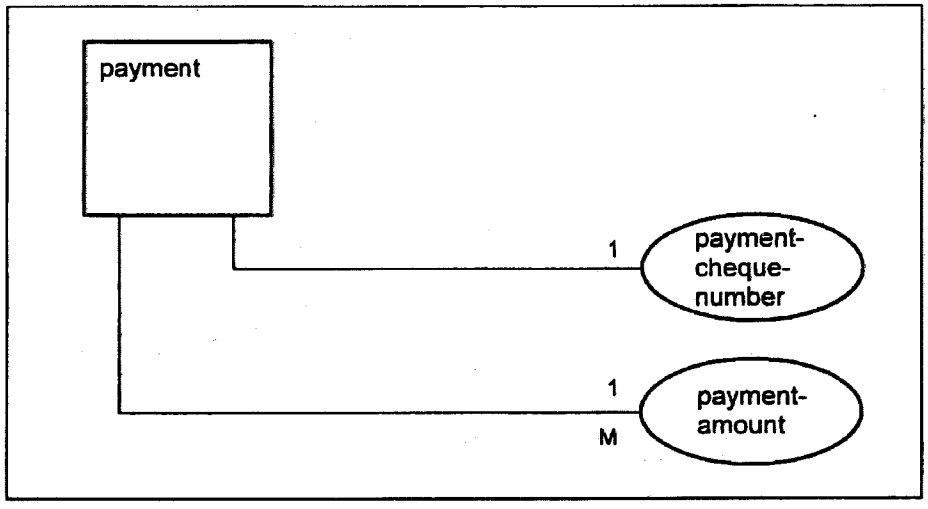


Figure App.1.5 Attributed Entity

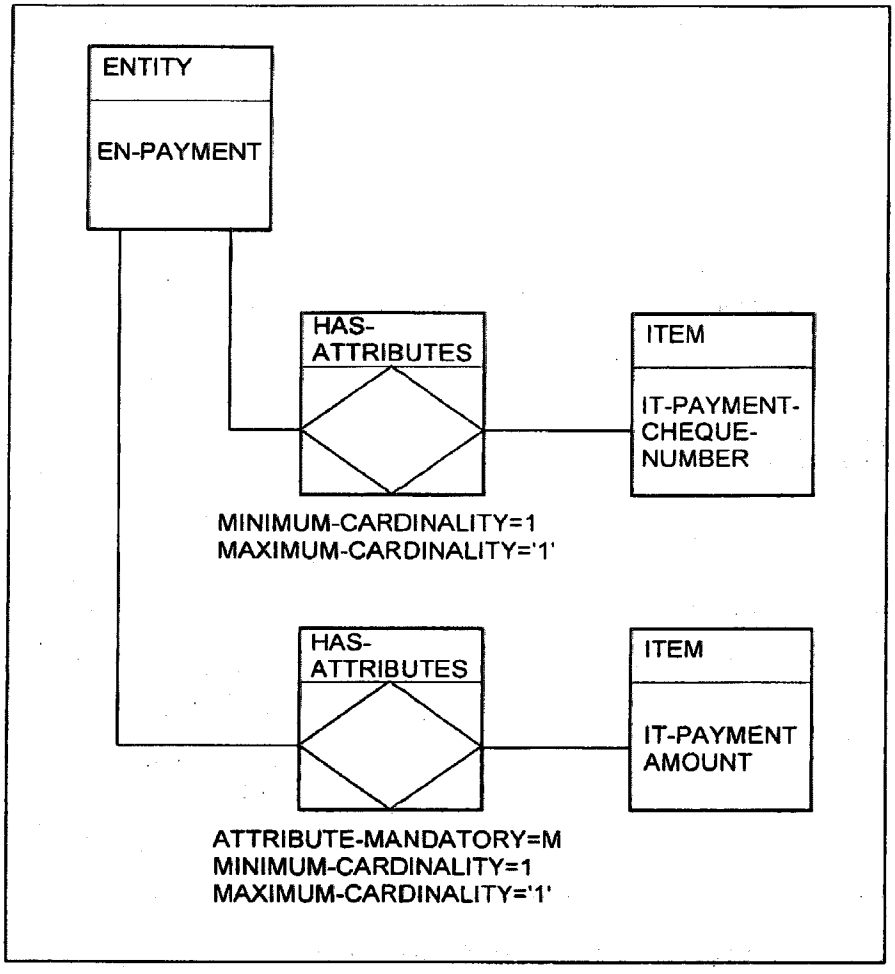


Figure App.1.6 Representation of Attributed Entity

App.1.4.4 Binary Business Relationship

A binary business relationship is shown in Figure App.1.7. The relationship "contains" relates the two entities "shipment-record" and "shipment-record-line". That is, "shipment-record" contains "shipment-record-line" and "shipment-record-line" is contained-by "shipment-record".

Note the following:

- the source cardinality of "contains" is 1 and the target cardinality is many, that is, an instance of "shipment-record" contains an unspecified number of instances of "shipment-record-line", but an instance of "shipment-record-line" is contained-by one instance of "shipment-record"

- the target of "contains" is mandatory, that is, an instance of "shipment-record-line" must be contained-by an instance of "shipment-record"

- the target of "contains" is controlled, that is, when the last valid instance of "contains" is deleted the target instance of "shipment-record-line" is also deleted.

This is represented in the repository as shown in Figure App.1.8. EN-SHIPMENT-RECORD is the source of the relationship and EN-SHIPMENT-RECORD-LINE its target. The equivalent member definitions are given below:

```
REPLACE EN-SHIPMENT-RECORD.
ENTITY
.
REPLACE X001.
BUSINESS-RELATIONSHIP
SOURCE EN-SHIPMENT-RECORD
TARGET EN-SHIPMENT-RECORD-LINE
FORWARD-VERB CONTAINS
INVERSE-VERB CONTAINED-BY
SOURCE-MINIMUM-CARDINALITY 1
SOURCE-MAXIMUM-CARDINALITY '1'
TARGET-MINIMUM-CARDINALITY 1
TARGET-MAXIMUM-CARDINALITY 'm'
SOURCE-MANDATORY O
TARGET-MANDATORY M
SOURCE-CONTROLLED U
TARGET-CONTROLLED C
SOURCE-DEPENDENT I
TARGET-DEPENDENT D
.
REPLACE EN-SHIPMENT-RECORD-LINE.
ENTITY
.
```

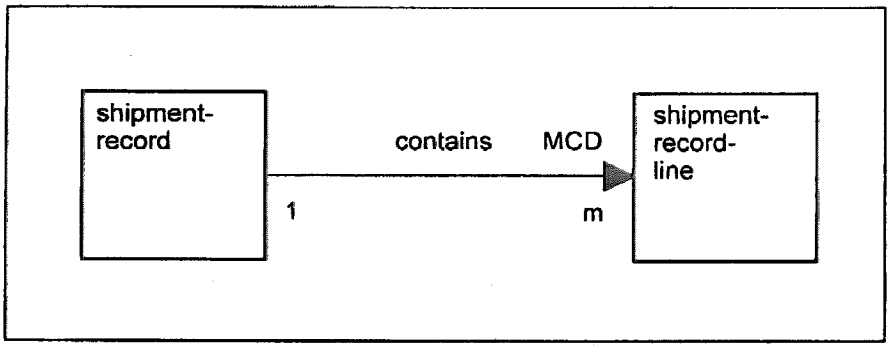


Figure App.1.7 Binary Business Relationship

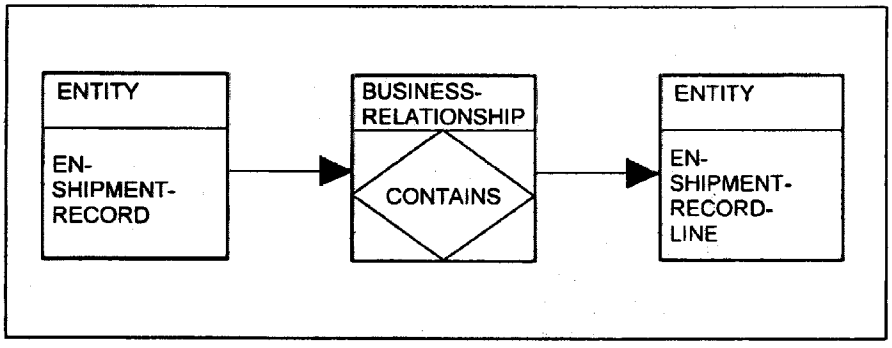


Figure App.1.8 Representation of Binary Business Relationship

App.1.4.5 Attributed Business Relationship

An attributed business relationship is shown in Figure App.1.9. The relationship "supplies" has a single attribute "minimum-supply-quantity". This is represented in the repository as shown in Figure App.1.10. The equivalent member definition is given below:

```
REPLACE EN-SUPPLIER.
ENTITY
.
REPLACE X002.
BUSINESS-RELATIONSHIP
SOURCE EN-SUPPLIER
TARGET EN-RAW-MATERIAL
FORWARD-VERB SUPPLIES
.
REPLACE EN-RAW-MATERIAL.
ENTITY
.
REPLACE X001.
HAS-ATTRIBUTES
SOURCE X002
TARGET IT-MINIMUM-SUPPLY-QUANTTIY
MAXIMUM-CARDINALITY '1'
MINIMUM-CARDINALITY 1
.
REPLACE IT-MINIMUM-SUPPLY-QUANTITY.
ITEM
.
```

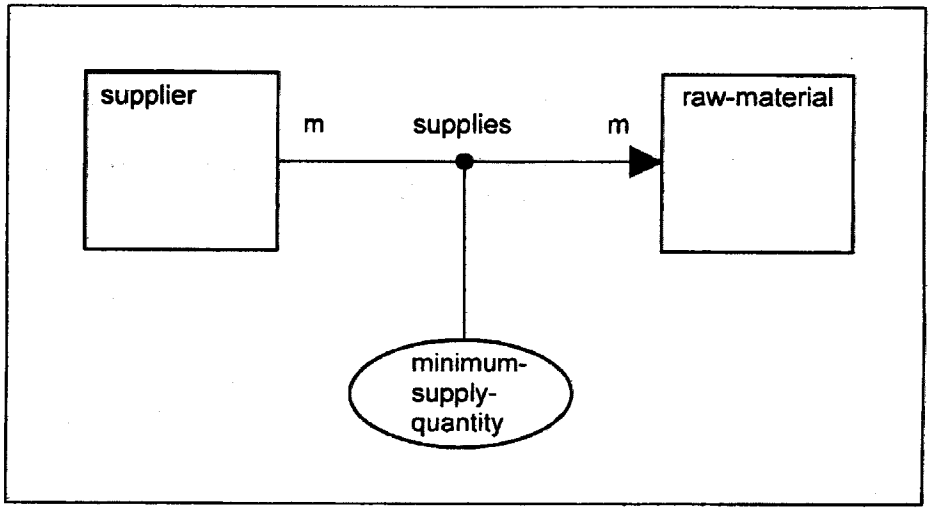


Figure App.1.9 Attributed Business Relationship

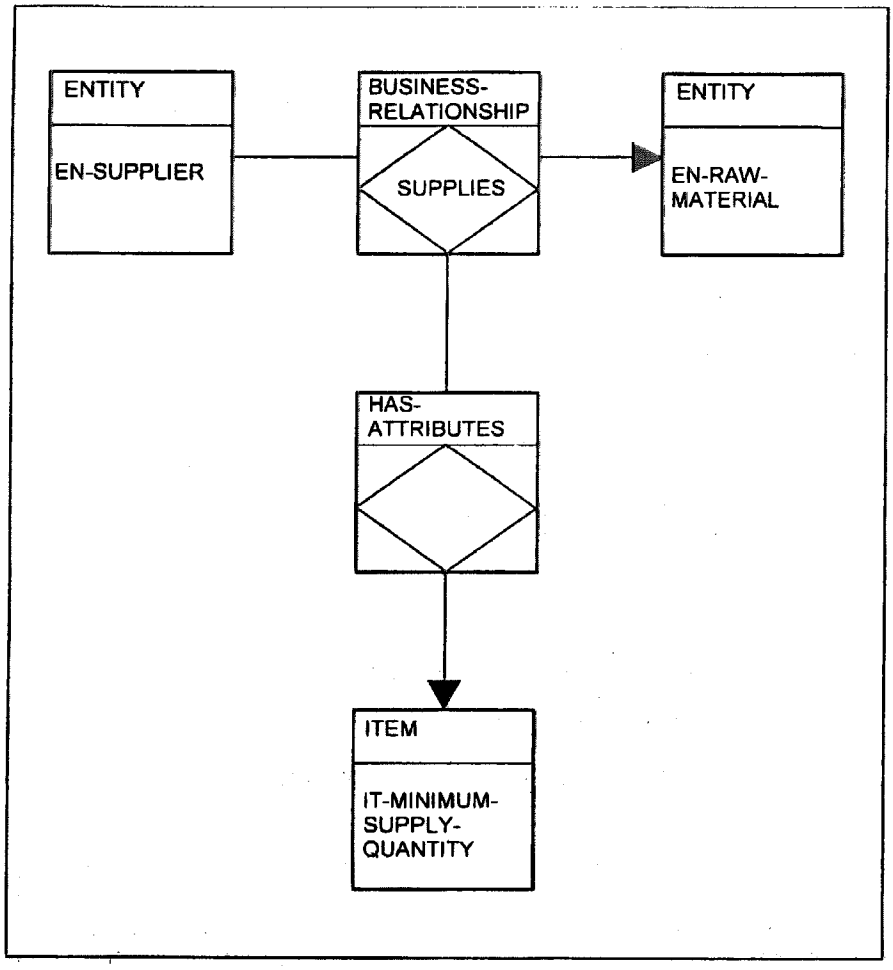


Figure App.1.10 Representation of Attributed Business Relationship

App.1.4.6 Business Relationship on a Business Relationship

A business relationship on a business relationship is shown in Figure App.1.11. The relationship "supplies" has an entity "supplier-price-group" associated with it via the relationship "priced-by". That is, a raw material supplied-by a supplier is priced-by a supplier-price-group.

This is represented in the repository as shown in Figure App.1.12. The equivalent member definitions are given below:

```
REPLACE EN-SUPPLIER.
ENTITY
.
REPLACE X001.
BUSINESS-RELATIONSHIP
SOURCE EN-SUPPLIER
TARGET EN-RAW-MATERIAL
FORWARD-VERB SUPPLIES
INVERSE-VERB SUPPLIED-BY
.
REPLACE EN-RAW-MATERIAL.
ENTITY
.
REPLACE X002.
BUSINESS-RELATIONSHIP
SOURCE X001
TARGET EN-SUPPLIER-PRICE-GROUP
FORWARD-VERB PRICED-BY
INVERSE-VERB PRICES
SOURCE-MINIMUM-CARDINALITY 1
SOURCE-MAXIMUM-CARDINALITY '1'
TARGET-MANDATORY M
.
REPLACE EN-SUPPLIER-PRICE-GROUP.
ENTITY
.
```

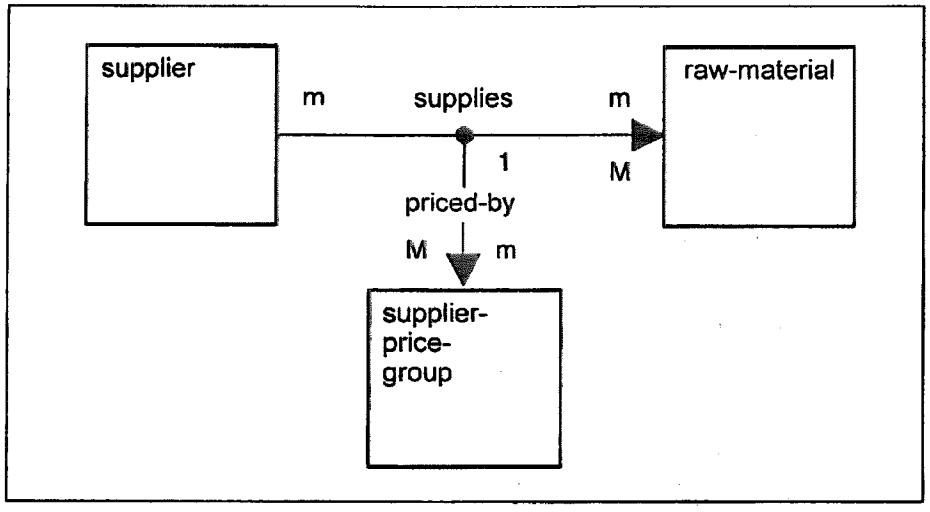


Figure App.1.11 Business Relationship on a Business Relationship

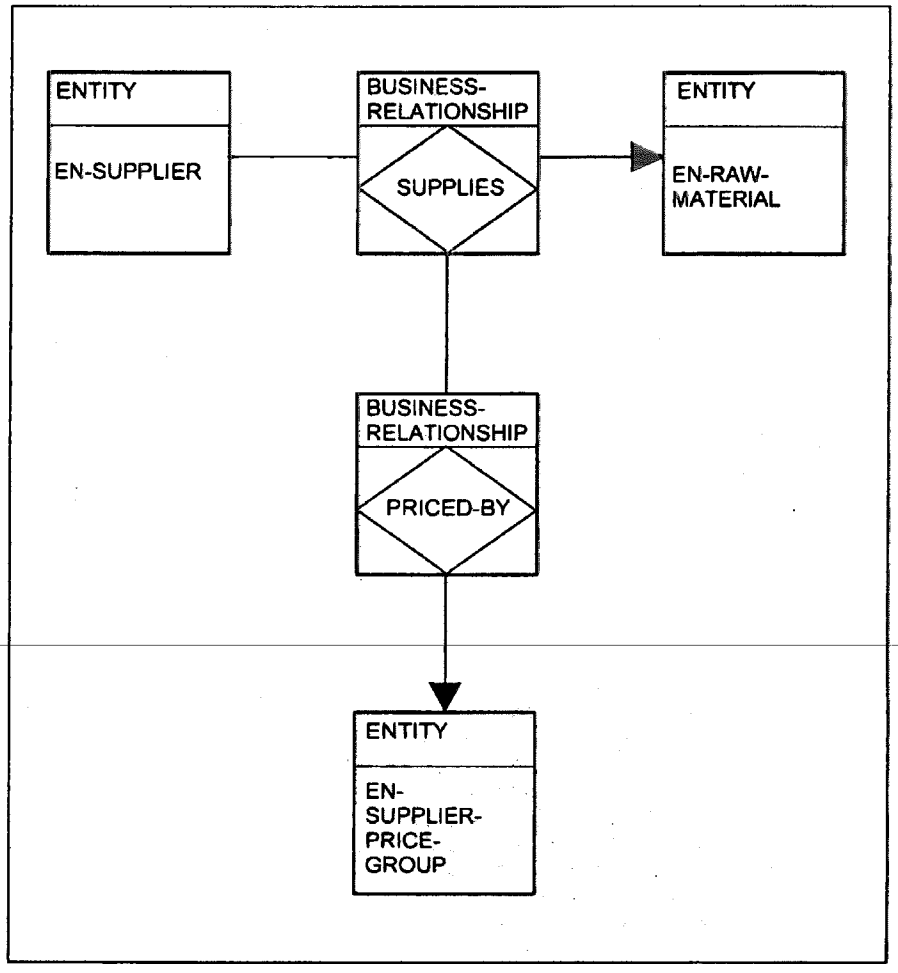


Figure App.1.12 Representation of Business Relationship on a Business Relationship

App.1.4.7 Existential Dependency

Existential dependency occurs when an entity depends for its existence on one or more other entities, through relationships it has with these.

A **weak entity** is an entity that cannot exist on its own and therefore cannot identify itself completely. A relationship can also be weak.

In practice weakness or dependency is not really a property of an entity or relationship. It is better defined as a property of relationship participation. For example, an entity is not weak absolutely, but only with respect to certain other entities or relationships.

Figure App.1.13 shows a weak entity "shipment-record-line". "shipment-record-line" is weak with respect to the entity "shipment-record". You would never refer merely to an instance of "shipment-record-line", only to an instance contained-by a given instance of "shipment-record".

This is represented in the repository as shown in Figure App.1.14. The equivalent member definitions are given below:

```
REPLACE EN-SHIPMENT-RECORD.
ENTITY
.
REPLACE X001.
BUSINESS-RELATIONSHIP
SOURCE EN-SHIPMENT-RECORD
TARGET EN-SHIPMENT-RECORD-LINE
FORWARD-VERB CONTAINS
INVERSE-VERB CONTAINED-BY
SOURCE-MAXIMUM-CARDINALITY '1'
SOURCE-MINIMUM-CARDINALITY 1
TARGET-DEPENDENT D
TARGET-MANDATORY M
TARGET-CONTROLLED C
.
REPLACE EN-SHIPMENT-RECORD-LINE.
ENTITY
.
```

The TARGET-DEPENDENT clause of X001 specifies that the target EN-SHIPMENT-RECORD-LINE is dependent on the source EN-SHIPMENT-RECORD.

An **N-ary relationship** is a relationship in which three or more entities co-participate. It can be represented by an entity that is weak with respect to all the entities participating in the N-ary relationship.

An N-ary relationship is shown in Figure App.1.15. The entities "doctor", "patient" and "nurse" participate in a three-way relationship "operation". This is represented in the repository as shown in Figure App.1.16. An extra ENTITY member EN-OPERATION has been introduced that, since it is weak with respect to EN-DOCTOR, EN-PATIENT, and EN-NURSE, exactly represents the N-ary relationship.

The equivalent member definitions are given below:

```
REPLACE EN-DOCTOR.
ENTITY
.
REPLACE EN-PATIENT.
ENTITY
.
REPLACE EN-NURSE.
ENTITY
.
REPLACE EN-OPERATION.
ENTITY
.
REPLACE X001.
BUSINESS-RELATIONSHIP
SOURCE EN-PATIENT
TARGET EN-OPERATION
TARGET-DEPENDENT D
.
REPLACE X002.
BUSINESS-RELATIONSHIP
SOURCE EN-DOCTOR
TARGET EN-OPERATION
TARGET-DEPENDENT D
.
REPLACE X003.
BUSINESS-RELATIONSHIP
SOURCE EN-NURSE
TARGET EN-OPERATION
TARGET-DEPENDENT D
.
```

The verbs of the BUSINESS-RELATIONSHIP members are here of no significance and are omitted.

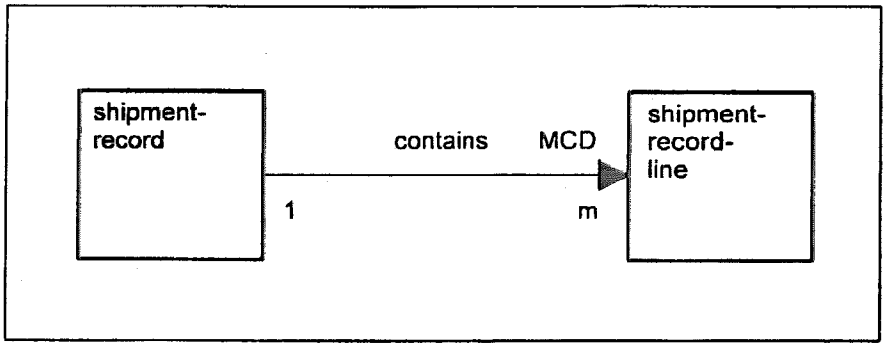


Figure App.1.13 Weak Entity

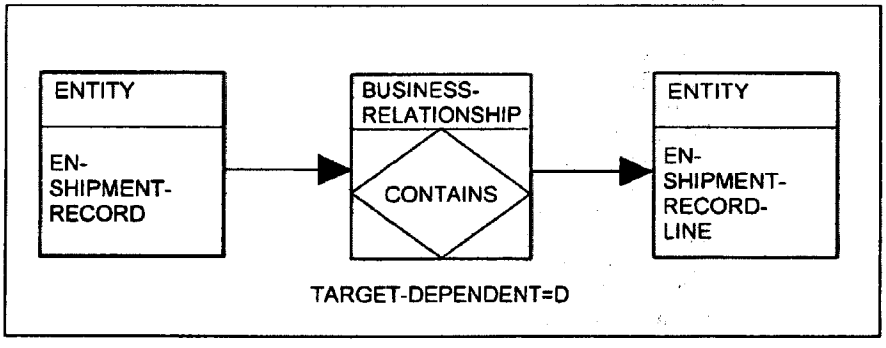


Figure App.1.14 Representation of Weak Entity

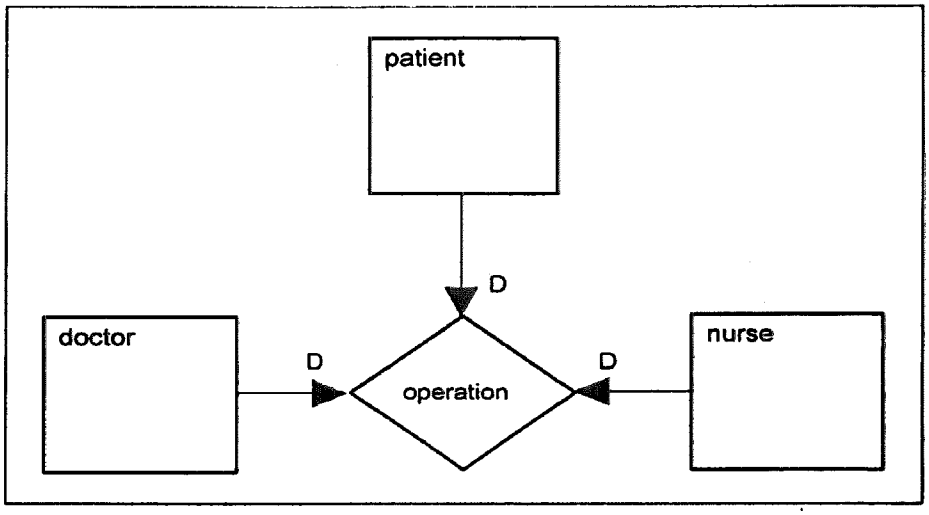


Figure App.1.15 N-ary Relationship

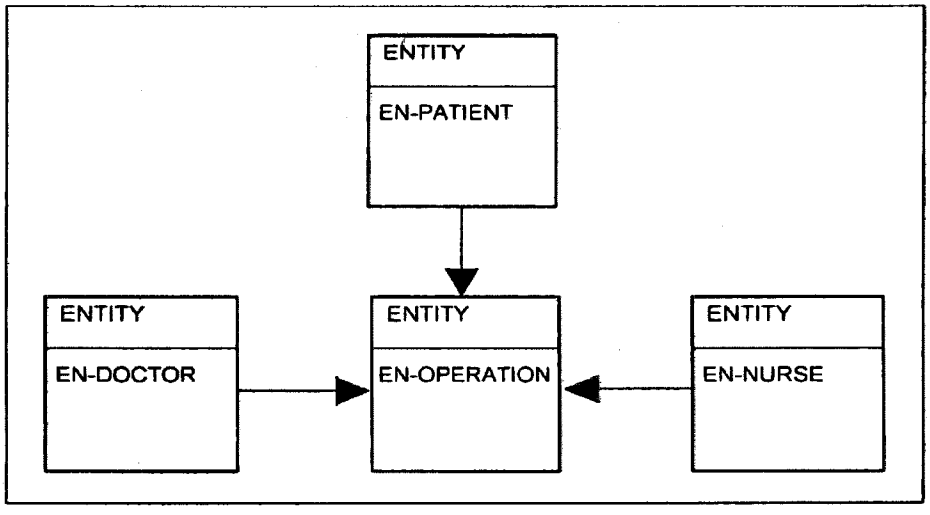


Figure App.1.16 Representation of N-ary Relationship

APP.2 COMMON CLAUSES AND ATTRIBUTES

APP.2.1 INTRODUCTION

Common clauses are clauses that can occur in any member definition statement; that is, they are common to all member types. The common clauses are all optional and can appear in any order in the member definition. Only one of each type of clause may be included in a member definition. Note that you may define up to nine UDR clauses (UDR1 to UDR9) in any one member.

The following common clauses can be defined in any MANAGER Products member type:

- ACCESS-AUTHORITY
- ADMINISTRATIVE-DATA
- ALIAS
- CATALOGUE
- COMMENT
- COPYRIGHT
- DESCRIPTION
- EFFECTIVE-DATE
- FREQUENCY
- NOTE
- OBSOLETE-DATE
- QUERY
- SEE
- SECURITY-CLASSIFICATION
- UDRn (where n is an integer of 1 to 9)

For the definition of the clauses common to member types within any UDS table, refer to section App.2.2.

In addition to the above common clauses, you can optionally define the following common attributes:

- DEFINITION-RESPONSIBILITY
- STANDING

in any of the member types defined in MDRIM (UDS table DU016).

For the definition of the common attributes, refer to section App.2.3.

The following attributes can be defined in any of the IEW member types defined in the MDRIM (that is, those member types whose names are prefixed with IEW).

- CREATION-DATE
- CONTRACTED
- LAST-SAVE-DATE
- LAST-UPDATE
- RELEASE-CODE
- RELATIVE-POSITION
- TOKEN
- POLICY-MAKER
- CHECKED-OUT
- UDR1 (IEW-ADW-XREF)
- UDR2 (IEW-ADW-XREF-DSTRUCT)

For the definition of the attributes common to IEW member types defined in the UDS table DU016, refer to section App.2.4.

APP.2.2 COMMON CLAUSE DEFINITIONS

There are six common clauses that enable you to include text in member definition statements. The clauses have the same syntax, but are used to store different kinds of information.

The format of these common clauses is as follows:

```
ADMINISTRATIVE-DATA 'text'
COMMENT 'text'
COPYRIGHT 'text'
DESCRIPTION 'text'
NOTE 'text'
QUERY 'text'
```

text is up to 32767 delimited character strings, each string having a maximum of 252 characters, including the delimiters.

The definitions of the common clauses follows:

ADMINISTRATIVE-DATA

This clause is used to record any useful information about the member being defined, such as the software release and any modifications made to the clauses.

COMMENT

May contain a text string of up to 32767 delimited character strings, each up to 252 characters long. It should contain any information you wish to record about the member.

COPYRIGHT

By defining the COPYRIGHT clause you have the ability to specify Copyright information.

DESCRIPTION

Contains a text string of up to 32767 delimited character strings, each up to 252 characters long. It should describe the member fully.

NOTE

Contains a text string of up to 32767 delimited character strings, each up to 252 characters long. It should contain any information you wish to record except administrative data.

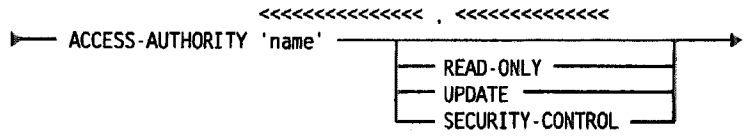
QUERY

May contain a text string of up to 32767 delimited character strings, each up to 252 characters long. It records any unfinished details about this member which need to be reviewed in the future.

ACCESS-AUTHORITY

Contains the name of one or more users, and the type of access authorized for each user. The three types of access authority are READ-ONLY, UPDATE, SECURITY-CONTROL. There is no default value for the clause.

The syntax of the clause is as follows:



where **name** is a name not more than 32 characters in length, conforming to the rules for user names and owner names.

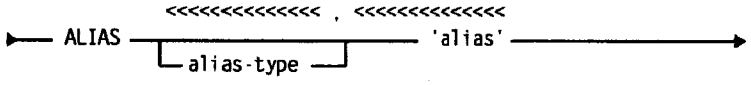
ALIAS

Assigns alternative names to a member being defined. Two of the types of alias available are:

- LONG, to assign an expanded and more descriptive name than the member's name

- SHORT, to assign a name that is shorter than the member's name.

The syntax of the clause follows:



where

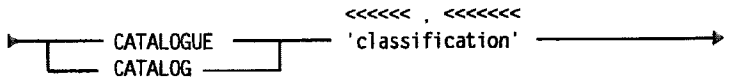
alias is the alternative name for the member and must not be more than 79 printable characters

alias-type is one of the specific alias-type keywords available in your repository. Use the SHOW ALIAS-TYPES command to find out the alias-type keywords available in your repository.

CATALOGUE

Contains any classification useful for indexing, grouping or extracting members, such as member types or a system name. Classifications must be within single quotes, separated by commas.

The syntax of the clause is as follows:

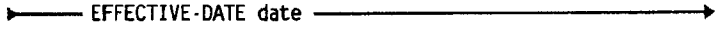


where **classification** is a string of not more than 79 characters.

EFFECTIVE-DATE

The date on which this member definition takes effect. It should be in the standard format your organization uses for dates.

The syntax of the clause is as follows:



where **date** must be in the format which applies in your installation.

OBSOLETE-DATE

Contains the date on which this definition became, or will become, obsolete, in your organization's standard date format.

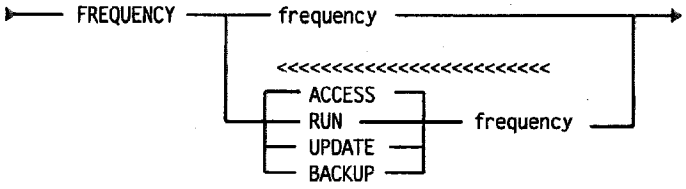
The syntax of the clause is as follows:

where **date** is as defined above.

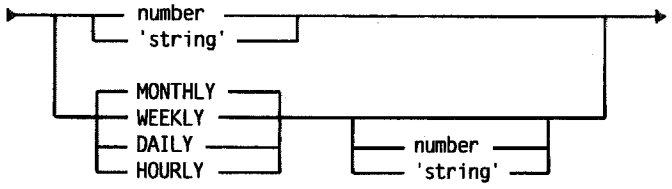
FREQUENCY

Documents the frequency with which the data or process described is accessed, run, updated or backed up. It may contain one frequency value after the keyword, or any number of subordinate clauses with frequency values. Subordinate clauses are separated by commas.

The syntax of the clause is as follows:



where **frequency** is:

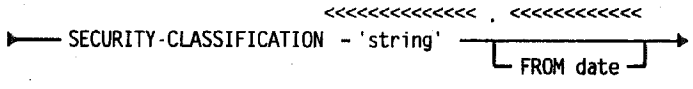


where

number is an integer
string is a string of not more than 256 characters.

SECURITY-CLASSIFICATION

Documents the security classification of the information in the member. FROM specifies the date from which the classification became effective. It may be superseded by a later FROM clause.



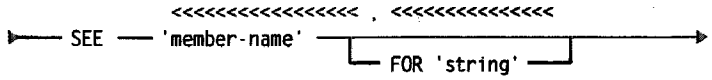
where

string is as defined above
date must be in the format which applies in your installation.

SEE

Allows you to create relationships between individual repository members.

The syntax of the clause is as follows:



where

member-name is the name of a repository member
string is as defined above.

UDR

The UDR clause allows you to define up to nine types of user-defined EA relationship types between member types, in addition to the pre-defined relationship types. You can specify UDR clauses in any of the member types available to you, including user defined member types. You can define up to nine UDRs in any one member, and each UDR may refer to a number of other members. You can also define a sub-relationship for each UDR. UDR clauses can optionally be qualified with a text sub-clause, in which you can provide information about the relationship.

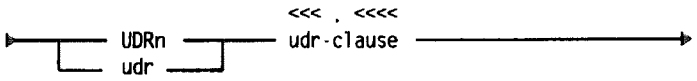
The repository Controller is responsible for defining the UDRs in a repository. To find out the names of the UDRs in your repository, enter the command:

```
SHOW MEMBER-TYPE VARIABLES FOR COMMON CLAUSES ;
```

To find out the relationships which are available for a particular member type, enter:

```
SHOW UDS MEMBER-TYPES RELATIONSHIPS FOR member-type ;
```

The syntax of the UDR clause is as follows:

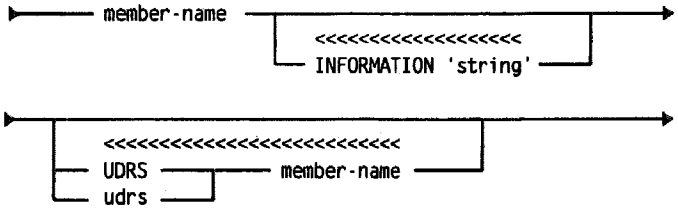


where

n is an integer of 1 to 9

udr is the name of a user defined relationship

udr-clause is:



where

member-name is the name of a repository member

string is a string of not more than 255 characters

udrs is the name of a user defined sub-relationship.

If the members you specify are not already encoded, they are created as dummy members of the same member type as the member containing the UDR clause.

For example:

```
UPDATED-BY DEPARTMENT-NUMBER INFORMATION 'update data'
                    UPDATE-PROGRAM PERSONNEL-UPDATE-6
```

In this example, the member being defined has a user defined relationship, UPDATED-BY, with the member 'DEPARTMENT-NUMBER' and a sub-relationship, UPDATE-PROGRAM, with the member 'PERSONNEL-UPDATE-6'.

User defined relationships and sub-relationships can be fully reported, interrogated and exported by the repository reporting, interrogation and export facilities.

For example, to find out ITEMs which have an "UPDATED-BY" relationship with the member, PERSONNEL-UPDATE-6, enter:

```
WHICH ITEMS USE PERSONNEL-UPDATE-6 VIA UPDATED-BY ;
```

To find out which FILEs, GROUPs or ITEMs have an "UPDATED-BY" relationship with any other repository members, enter:

```
GLOSSARY SIGNIFICANT FILES, GROUPS, ITEMS GIVING
                                    UPDATED-BY ;
```

You can retrieve UDRs from a member which has been retrieved by a DACCESS command in an executive routine. For example, if you include the following command in the executive routine:

```
DRETRIEVE FIRST udr-name ;
```

the output would include the first occurrence of the UDR specified. (Use the SHOW MEMBER-TYPE command as above to find out the names of the UDRs in your repository.)

APP.2.3 COMMON ATTRIBUTES

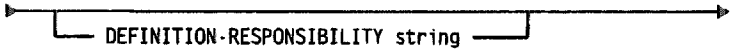
The following common attributes can be defined in any member type defined in the Dictionary/Repository Information Model:

- DEFINITION-RESPONSIBILITY
- STANDING

DEFINITION-RESPONSIBILITY

This clause may contain a text string. It is used to identify the person responsible for the definition of this member.

The syntax of the clause is as follows:

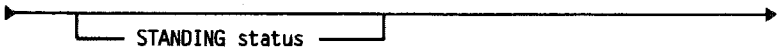


where **string** is a character string of no more than 79 characters.

STANDING

Describes the quality or current status of the member's definition.

The syntax of the clause is as follows:



where **status** is one of the following permitted values:

D for Draft
P for Proposed
A for Approved
S for Standardized

APP.2.4 IEW COMMON ATTRIBUTES

The following common attributes can be used in any of the IEW member types (those member types whose names are prefixed with IEW) defined in the UDS table DU016.

The definitions for these clauses follow:

CREATION-DATE

Date of creation of this object Format YYYY/MM/DD HH:MM:SS USRIDENT

CONTRACTED

An object can be either CONTRACTED or EXPANDED. A CONTRACTED object hides all the objects it references. This attribute is common to all IEW/ADW objects.

LAST-SAVE-DATE

Date of last save IEW format YYYY/MM/DD HH:MM USRIDENT

LAST-UPDATE

Date of last update Format YYYY/MM/DD HH:MM USRIDENT

RELEASE-CODE

A character string indicating the relevant release.

RELATIVE-POSITION

Relative position of the object in the diagram.

TOKEN

ADW /IEW internally generated unique identifier. An eleven digit number only meaningful within a single import or export.

POLICY-MAKER

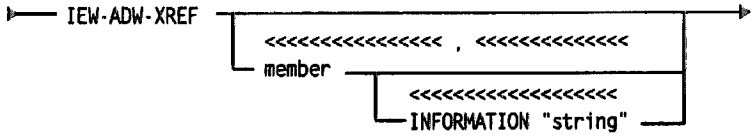
Policy maker.

CHECKED-OUT

ADW/MVS check-out encode/object tokens.

UDR1 (IEW-ADW-XREF)

The **udr1** clause is a user defined EA relationship type. By default, this is:



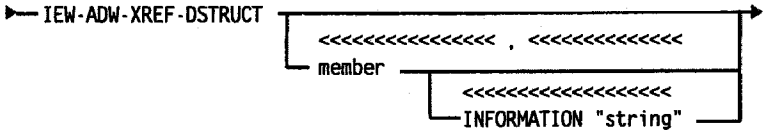
where

member is the name of a repository member

"string" is the data to be imported from, or exported to, ADW/IEW.

UDR2 (IEW-ADW-XREF-DSTRUCT)

The **udr2** clause is a user defined EA relationship type. By default, this is:



where

member is the name of a repository member

"string" is the data to be imported from, or exported to, ADW/IEW

AMENDMENT RECORD

This manual when issued should contain 650 pages (325 sheets), comprising:

Preliminary pages	14	pages	(7 sheets)
Chapter 1	4	pages	(2 sheets)
Chapter 2	592	pages	(296 sheets)
Appendix 1	26	pages	(13 sheets)
Appendix 2	12	pages	(6 sheets)
Amendment Record	2	pages	(1 sheet)

(Blank pages occurring at the end of chapters or appendices are included in the page counts.)

Please check that all pages are present immediately on receipt. If any pages are missing, please inform your local MSP Product Supplier without delay.

If any updates to the information contained in this manual become necessary, additional or replacement pages will be issued. Such pages will be accompanied by a front sheet, the amendment list, detailing the action required to update the manual. On completion of the updating, the amendment list should be filed behind this page. An entry should be made in the table overleaf to record the action taken.

Each page of this manual is annotated (immediately above the page reference) with the publication date of the page. Update pages will be similarly annotated. It is thus always possible to check that the latest version of a page is held.

Amendment Record

Amendment List No.	Date of Incorporation	Initials	Notes
1	980916	NMD	

MSP Amendment List

METHODMANAGER

Dictionary/Repository Information Model

(Second Edition)

Amendment List 1 Date of Issue: 03.95

In the event of any query on this amendment list, please telephone your MSP Product Supplier at the number given on the card at the front of your MANAGER Products binder.

© Copyright Management Systems & Programming Limited (MSP Ltd) and/or WHDL - MSP INC 1995 All Rights Reserved. Licensed Material - Program Product & Data Structures Confidential Property of MSP Ltd and/or WHDL - MSP INC

The amendments listed below should be made to the above-named publication.

On re-issued pages, information that has changed substantially since previous editions is indicated by vertical lines in the margins. When the page is next re-issued, these vertical lines will be removed.

1	Remove and destroy pages:	Insert the new pages:
	i to iv	i to iv
	2-27, 2-28	2-27, 2-28
	2-31, 2-32	2-31, 2-32
	2-233, 2-234	2-233 to 2-234.2
	2-249, 2-250	2-249 to 2-250.2
	-	2-256.1 to 2-256.4
	2-263, 2-264	2-263 to 2-264.18
	-	2-276.1 to 2-276.2

Remove and destroy pages:	Insert the new pages:
2-279, 2-280	2-279 to 2-280.2
2-291 to 2-296	2-291 to 2-296.2
2-299, 2-300	2-299 to 2-300.2
-	2-316.1, 2-316.2
2-323, 2-324	2-323, 2-324.2
2-339, 2-340	2-339, 2-340.4
2-363 to 2-368	2-363 to 2-368
3-371, 3-372	3-371 to 3-372.2
-	2-382.1, 2-382.2
3-383 to 3-392	3-383 to 3-392.6
App.2-1 to App.2-12	App.2-1 to App.2-12

2 Update page AR-2 to record the incorporation of Amendment List 1.

3 File this page behind page AR-2.