software AG

Predict

Predict and Other Systems

Version 8.5.1

October 2021

ADABAS & NATURAL

Table of Contents

Predict and Other Systems

Predict supports a wide variety of application development environments, database management systems and programming languages. Many functions support the active use of data stored in the dictionary when developing applications and when using these applications in a production environment.

This documentation describes how Predict is used with specific systems or facilities.

Verifications and Processing Rules	The interaction between verification objects in Predict and processing rules in Natural.
Steplib Support	Provides an overview of the areas in Predict affected by the Steplib concept in Natural. documentation.
Adabas Vista	How to define distributed data structures for working with Adabas Vista in Predict, and how to generate from these definitions the objects you need for the physical implementation of these structures.
VSAM	How to document physical and logical VSAM structures in Predict; how to generate DDMs from VSAM objects in Predict; using Natural for VSAM with physical VSAM; using a record layout concept.
Natural for DL/I	How to document IMS/DL/I data structures; how to create objects for IMS/DL/I with incorporation functions; how to maintain documentation for IMS/DL/I; generation functions for files of types I, J and K.
DB2 and SQL/DS	How to document DB2 objects in Predict; generating, incorporating comparing and administering DB2 objects.
Oracle	How to document Oracle objects in Predict; generating, incorporating comparing and administering Oracle objects.
Static SQL	Describes how to document the use of Static SQL in Predict and how to generate DBRMs from the Predict documentation. Retrieval functions and consistency checking is discussed, along with a description of how to use Predict information when binding application plans.
Adabas D and Other SQL Systems	Predict support of the following DBMS: Adabas D, Oracle, Ingres, Informix, Sybase. How to document SQL objects in Predict; generating and incorporating SQL objects.
Adabas SQL Server	Describes how to document Adabas SQL Server objects in Predict. Generation, Incorporation, Comparison and Administration functions which process Adabas SQL Server objects are described. This section also discusses XRef data created for Adabas SQL Server.
Adabas SQL Gateway	Describes how to document Adabas SQL Gateway objects in Predict. Generation, Incorporation, Comparison and Administration functions which process Adabas SQL Gateway objects are described.

Third Generation Languages	Describes the documentation of 3GL applications and programs and how XRef data for these applications is generated and maintained. The Predict functions used when developing a 3GL application are explained here.
Predict and Natural Development Server	Describes the documentation of Natural Development Server objects.

1   About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *folder.subfolder.service*, APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **https://documentation.softwareag.com**.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.aspx** and give us a call.

Software AG Tech Community

You can find documentation and other technical information on the Software AG Tech Community website at **https://techcommunity.softwareag.com**. You can:

- Access product documentation, if you have Tech Community credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Verifications And Processing Rules

Terminology

Automatic Rule

An automatic rule is used automatically whenever a field to which a verification of status automatic has been linked via "Is verified by VE" is included in a map. Automatic rules cannot be changed in the Natural map editor, which guarantees consistent use of these processing rules throughout an application.

Automatic - Verification Status

A verification has the status automatic if it contains a rule that is linked to at least one field of at least one file for which a DDM has been generated.

Conceptual - Verification Status

A verification of status conceptual contains a rule that has not yet been cataloged with `CAT FREE` or `SA[VE] FREE`.

Documented - Verification Status

A verification of status documented does not contain a rule and cannot be used in a Natural map. It is used in the early phases of application design.

Free Rule

A free rule is used in a Natural map by specifying the ID of a verification of status free. The Natural code of the processing rule stored with the verification will be included into the map when the map is cataloged. Free rules can be defined and modified with the Natural map editor and directly in Predict.

Free - Verification Status

A verification has the status free if it contains a rule that has been cataloged with `CAT FREE` or `SA[VE] FREE`.

Inline Rule

Inline processing rules are defined within a Natural map source and do not have a name assigned. Inline rules in Natural can be used independently of Predict.

Natural Construct - Verification Status

A verification of status Natural Construct is created by entering command `CAT N` or `SA[VE] N` in the Rule Editor. These verifications are only used by Natural Construct.

Processing Rule

Rule for validating data entry in a Natural map or SQL database. The following types of processing rules can be defined:

For Natural:

- Inline processing rules

- Free rules

- Automatic rules

For SQL:

- SQL

Rule Editor

The Rule Editor is a modified Natural Editor in Predict which is used to edit the rule of a verification. See the section *Editors in Predict* in the *Predict Reference* documentation.

SQL - Verification Status

A verification of status SQL is created by entering command `CAT S` or `SA[VE] S` in the Rule Editor. See *Verifications of Status SQL*.

Verification

A verification is a predefined object type in Predict and can contain the Natural or SQL code of a processing rule.

General Information

Natural processing rules perform validity checks on input data to ensure that the data to be processed is suitable. For example, in a program controlling traffic lights, the only input values allowed for the field colour might be red, green and amber.

Natural processing rules can be defined and stored centrally in Predict.

Benefits

Storing processing rules in Predict has the following advantages:

- Programming costs and the number of errors can be reduced by using free rules stored in Predict.
- The use of processing rules can be forced by linking automatic rules to fields via association "Is verified by VE". Automatic rules cannot be changed in the Natural map editor. This guarantees consistent use of these processing rules throughout an application.
- The use and design of processing rules can be planned and revised by using verifications of status documented and conceptual.

Rules Applying to Processing Rules in Predict

The following general rules apply to processing rules in Predict:

- Processing rules of status documented or conceptual can be linked to fields via "Is verified by VE".
- Natural inline processing rules can be integrated into Predict as free rules by giving them a name in the Natural map editor.
- Automatic rules can be used as free rules by specifying their name in the map.

- Processing rules can be written in either structured mode or report mode.

- A `GENERATE` command creates the Natural code of a processing rule from the rule of a verification. The generated code can then be changed to meet specific requirements.

- Links from fields to verifications linked via "Is verified by VE" are rippled. See *Rippling Verifications*

Using Rules of Verifications in an External Environment

The status of the Predict verification object determines how a rule stored with the verification is used in an external environment. Six status types are distinguished:

- **Documented**
- **Conceptual**
- **Free**
- **Automatic**
- **Natural Construct**
- **SQL**

The characteristics of the different status types are described below.

Verifications of Status Documented

Verifications of status documented can be used in the early phases of application design when the parts of an application that have to be implemented are listed. verifications of status documented do not contain processing rules and cannot be used in Natural maps.

Documented	
ID:	xyz
Type:	equal
Format:	alpha
Value	'xyz'

Verifications of type documented are not connected to an external environment.

Verifications of Status Conceptual

Verifications containing a processing rule that has not yet been cataloged with `CAT FREE` or `SA[VE] FREE` have the status conceptual.

Verifications of Status Free

A verification has the status free if it contains a processing rule that has been cataloged with `CAT FREE` or `SA[VE] FREE` and can therefore be used in any Natural map by linking it explicitly to a field.
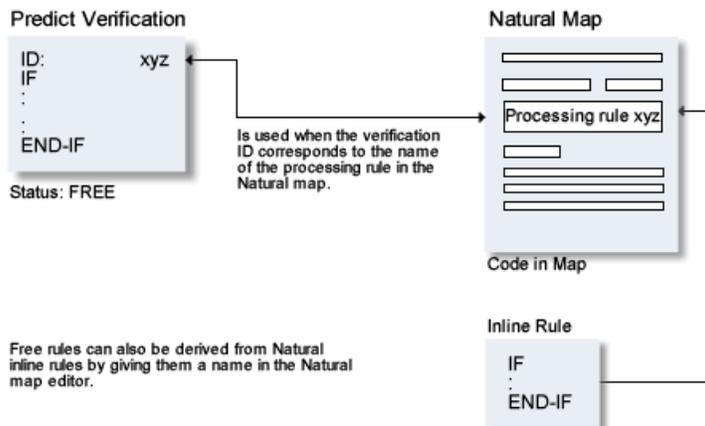
A free rule is used in a Natural map by specifying the verification ID. A Select function is provided for selecting free rules from Predict. Only free rules with a format compatible with the format of the input field of the map for which they are to be used will be displayed by the select function. The Natural code of the processing rules stored under the given verification ID will be included into the map when the map is cataloged.

Free rules can be defined and modified with the Natural map editor and directly in Predict.

Verifications of Status Automatic

A verification has the status automatic if it contains a processing rule that is linked to at least one field of at least one file for which a DDM has been generated. An automatic rule is automatically used every time a field to which it has been linked is included in a map. Automatic rules are centrally defined by the administrator who generates DDMs and cannot be modified by individual programmers with the Natural map editor. Defining an automatic rule is a two-stage process:

1. Link the verification containing the rule to a field of a file (a real file or a userview) in Predict.

2. Activate the rule by generating a DDM for that file.

Rules Applying to Automatic Rules

The following rules apply for the use of automatic rules:

■ Up to 50 automatic rules can be linked to a field.

■ If the code contained in an automatic rule is changed but its links to fields remain, the Predict Replace Verification Rule function can be used to update the active code which is used by the fields. There is no need to regenerate the DDM. Natural maps using processing rules that have been modified should be recataloged to ensure that they use the new version of the processing rule.

■ The LIST XREF function with the option Save set set to Y can be used to recatalog maps efficiently.

■ An automatic rule can also be used as a free rule by specifying the name of the corresponding verification in a Natural map. Automatic rules that have been used in this fashion cannot be modified with the Natural map editor.

■ Automatic rules cannot be changed with the Natural map editor.

Verifications of Status Natural Construct

Verifications of status Natural Construct can be accessed only from Natural Construct.

These verifications must be linked to a field so that Natural Construct can access them. Unlike verifications of status Automatic, it is not necessary to generate a DDM for a verification of status Natural Construct in order that the rule is used.
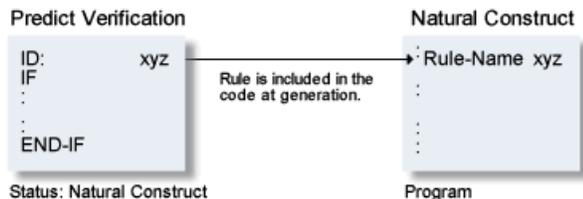
Verifications of Status SQL

The following rules apply:

■ Verifications of status SQL are skipped by functions Generate DDM and Replace Verification rule.

■ The syntax of the processing rules is not checked by Predict.

■ These processing rules may only contain references to the field to which they are linked. Ampersand notation is used instead of a fixed allocation.

■ The ampersand references are replaced by the corresponding field name when the `CREATE TABLE` statement is generated.

- If you execute the command `GEN[ERATE] S` in the Rule Editor, a corresponding SQL clause is created for all rule types except user routine. When the code of the rule is saved, the status of the verification is changed to SQL.

- Only one verification of status SQL (and 49 of a status other than SQL) may be linked to a field. This condition is checked only when the `CREATE TABLE` statement is generated.

- Comments are removed when the `CREATE TABLE` statement is generated.

- Otherwise the handling of these verifications corresponds to verifications of status Natural Construct. See *Verifications of Status Natural Construct*.

How Predict Stores Processing Rules

Processing rules are stored in Predict as attributes of verifications. If a verification has been linked to a field of a map, the Natural code of the rule is inserted when that map is cataloged.

Variable Names in Processing Rules

In the source code of a processing rule, the name of a variable can be represented by an ampersand (&). The Natural compiler or Predict generation function will substitute the name of the field (or PF key for a PF key rule) for the ampersand. This allows the use of a rule for different fields.

Example:

```
IF & = ' ' REINPUT 'ENTER NAME' MARK *&
```

Priority of Processing Rules

1. Processing rules assigned to function keys have highest priority.
2. Rules linked to different fields of a map are executed in the order in which the fields appear on a terminal screen.
3. A rank from 0 to 99 can be allocated to each inline rule or free rule linked to a field of that map. Additionally a rank can be allocated to all automatic rules linked to a field. The rules linked to that field will then be executed in ascending order of rank.
4. Automatic rules linked to a field in Predict are executed in the order their Predict verification IDs appear in the verification list of the field.

Processing Rules and Field Formats

Every rule is allocated a format to ensure that the rule will be compatible with the field format. The following table lists the compatible combinations of field format and rule format.

Field Format	Compatible Rule Format
A	A,B
N,P	N
I,F	B
B =< 4	A,B,N
B > 4	A,B
D (date/time)	D (date/time)
L	L

The rule format K (function key) applies exclusively to free rules.

Generating Processing Rules from Verifications

Processing rules can be created from Predict objects of type verification. Follow the steps below:

1. Enter Y in the field labelled Additional attributes in an Add, Copy or Modify Verification screen and select Rule code or execute the Edit rule of a Verification function (code R).

2. Enter the `GENERATE [S|N]` command in the Rule Editor to generate a first version of the processing rule from the definitions in the verification.

3. Modify the processing rule as required.

4. Test the rule with the `RUN` or `CHECK` command (Natural rules only).

5. The rule is cataloged/saved with either one of the following commands:

   - `SA[VE] [[FREE] RET[URN]]`
   - `CAT [[FREE] RET[URN]]`
   - `SA[VE] [S|N]`
   - `CAT [S|N]`

If `FREE` is used, the rule is stored as a free rule.

> **Note:** Commands `SAVE` or `CAT` do not perform a syntax check. The syntax is checked when you catalog a map that uses the rule or when the `CREATE TABLE` statement is executed.

Editing the Rule of a Verification

The rule of a verification is edited with the Predict Rule Editor. This editor can be invoked in one of the following ways:

- Enter Y in the field Additional attributes in the bottom line of the Add, Copy or Modify Verification screen and select Rule code.
- Call function Edit rule in the Verification Maintenance Menu.
- Enter command `EDIT VERIFICATION RULE <Verification-ID>`

Editor Commands

> **Note:** This section describes rule-specific editor commands. General editor commands are described in the section *Editors in Predict* in the *Predict Reference* documentation.

`CAT [[FREE] RET[URN]]`	Catalog/save the edited rule as a free rule. This command is only available when creating new rules and when editing conceptual rules.
`SA[VE] [[FREE] RET[URN]]`	**Note:** The commands `SAVE` and `CAT` do not perform a syntax check. The syntax is checked however, when you catalog a map that uses a rule.
`C[HECK]`	Checks whether the edited rule's Natural syntax is valid and reports errors.
`GEN[ERATE]`	Generates a processing rule from the values defined in the rule of the verification and adds it to the end of the Natural source in the rule editor. This command is not available for verifications of type User routine.

A table which shows the Natural and SQL statements generated for the different verification types can be found in the section *Rule Editor* in the *Predict Reference* documentation. |
`GEN[ERATE] N`	Generates a rule for Natural Construct from a verification of status documented (D). The status of the verification is changed to N.
`GEN[ERATE] S`	Generates an SQL clause for all verification types except user routine. When the code is saved, the status of the verification is changed to S.
`GLOBALS SM=OFF`	Switch to the reporting mode of Natural.
`GLOBALS SM=ON`	Switch to the structured mode of Natural.
`RENUM[BER], N`	Renumber the source lines in steps of N and renumber references to them accordingly.
`RUN`	Checks the edited rule. If no errors are found, a map is produced with which the user can test the rule by entering input values. The following rules apply:

- Length and format of the input field are derived from the rule format. For rules with format A, B or N, an additional window is displayed, where the derived field length can be overwritten. |

Rule Format	Format of the derived field	Length of the derived field
A	A	66
B	B	33
D	D	
L	L	1
N	N	27

- `RUN` tests a rule of format K (function key) without input data.
- For a rule of format L (logical), a blank space means false and any other input value means true.
- The stack must not be changed.
- The contents of the source area must not be changed.

  Note: All variables used except the ampersand (&) must be defined within the code.

- The variable names `SYSDIC-C1` and `SYSDIC-C2` are used for internal purposes and must not be used within the rule.
- The source will be renumbered.

Changing the Status of a Verification

Predict assigns the status of verifications itself. The following table shows which actions cause a change of status.

Old Status	New Status	Action
inline	free	Give the rule a name in the map editor.
documented	conceptual	Add a rule to the verification.
conceptual	free	Either catalog the rule in the rule editor with the command `SAVE FREE` or `CAT FREE` or use the Rename Verification function to change the status explicitly.
conceptual	SQL, Natural Construct	Catalog the rule in the rule editor with the command `SAVE S|N` or `CAT S|N`. At least one line must have been changed before cataloging.
free	inline	Change the rule's name to a blank in the map editor. The rule will still exist in Predict with status free.
conceptual	automatic	Link the rule to at least one field (with the field maintenance function Link Verification), then generate a DDM for the file which includes it.

Old Status	New Status	Action
free	automatic	Link the rule to at least one field (with the field maintenance function Link verification), then generate a DDM for the file which includes it.
free	conceptual, Natural Construct	Use the Rename Verification function. The status of a free rule cannot be changed to conceptual if the rule is used in any Natural map.
automatic	conceptual	Unlink all fields from the rule (with the field maintenance function Link Verification) then regenerate the related DDMs. If the rule is also used as a free rule, the status of the verification will be changed to free.
documented	SQL, Natural Construct	Generate a rule for Natural Construct from a verification of status documented (D) using the GEN[ERATE] N command of the Rule Editor; for SQL using the command GEN[ERATE] S.
Natural Construct	free, conceptual	Use the Rename Verification function.

Example

One of six town names is allowed as input. The verification describing this validity check is created with values as shown below:

```
10:13:40                 *****  P R E D I C T  *****                 2007-05-31
                            - Modify Verification -
Verification ID . TEST-TOWN                        Modified 2007-05-31 at 09:46
Status .......... Free                                        by HNO
Keys ..                                                            Zoom: N

Format .........* A  Alphanumeric                     Modifier    Zoom: N
Type ...........* T  Table of values
Message nr ......
Replacement 1 ...
Replacement 2 ...
Replacement 3 ...
Message text .... No SAG-office in that town.

Abstract    Zoom: N              Values    Zoom: N
                                    BRUESSEL
                                    RESTON
                                    PARIS
                                    DERBY
                                    CAMBRIDGE
                                    DARMSTADT


   Additional attributes ..* N         Associations ..* N
```

The following processing rule is generated if the GENERATE command in the Rule Editor is applied to this verification.

```
* ******************************************************************
* Verification: TEST-TOWN generated by PREDICT                    *
*  with format: Alphanumeric; Type: Table of values;             *
*          on: 2007-05-31; at: 10:13:33; from user: HNO          *
* ******************************************************************
IF NOT ( & = 'BRUESSEL' OR = 'RESTON' OR = 'PARIS' OR = 'DERBY'
OR = 'CAMBRIDGE' OR = 'DARMSTADT' )
      REINPUT 'No SAG-office in that town.'
         MARK *&
```

Rippling Verifications

Rippling Verifications from Standard Files

Each field of a standard file can have a list of verifications via association "Is verified by VE", which apply to that field. When the list is edited, corresponding changes are automatically made in the verification list of every field related to that standard field, according to the following rules:

- Every verification contained in the verification list of a standard field must also be contained in the verification list of a field related to that standard field. However, the sequence of verifications in the lists can differ.

- If a verification ID is changed, the same change is automatically made to that verification ID everywhere it appears in a verification list of related fields.

- If a verification ID is deleted, every instance of that verification ID is automatically deleted from the verification list of every related field.

- If a verification ID is added anywhere in the list, the same verification ID is automatically added to the end of the verification list of every related field.

- A verification ID can be removed from verification lists of related fields that are marked as no check against standard.

Rippling Verifications from Physical Files to Userviews

Fields of physical files can have verifications linked to them via "Is verified by VE". When a list of verifications linked to a field in a physical file is modified, corresponding changes are automatically made in the verification list of userview fields related to that field in the file. The following rules apply:

- The verification list of a field in a userview does not have to contain all the verifications that are contained in the list of the physical file field from which the userview field has been related. Hence, verifications can be deleted from the verification lists of userview fields, after these have been related to physical files.

- If a verification ID is changed, the verification ID is changed in the verification lists of all related fields.

- If a verification ID is deleted, every instance of that verification ID is automatically deleted from the verification list of every related field.

- If a verification ID is added, it is added to the verification lists of related fields.

3 Steplib Support

Natural as well as 3GL applications allow up to 8 steplibs for one main library. This structure can be documented in Predict with the object type library structure. This structure is evaluated by LIST XREF and active retrieval functions for programs and systems.

General Information

Predict supports the Natural steplib concept using the features listed below.

Object Type Library Structure

An object of type library structure documents a structure which describes a runtime or development environment (for example libraries for copy code). The system objects which document these libraries are linked as children to the library structure via "Contains SY". The following rules apply:

- The first entry in the link list is the main library, the following entries are steplibs.
- The link list of a library structure can contain up to 10 systems of type A (Application):
  - the first system in the list is the main library
  - the default steplib *STEPLIB plus up to 8 additional steplibs can be defined.
- The link list can contain additional systems of type G (3GL Application), but the maximum number of linked systems is 15.
- Dummy objects and systems without an implementation pointer for Library are permitted in the link list, but these objects are ignored when the library structure is evaluated for active retrieval function Program using programs and all LIST XREF functions.

See the section *Library Structure* in the *Predefined Object Types in Predict* documentation.

Program Type Dynamic

Programs of type dynamic are used to document calls of programs of the same name from different steplibs depending on the library structure. The following rules apply:

- Because programs of type dynamic document any number of implemented members, no check is performed as to whether the members documented by the program are actually implemented.
- With the active retrieval function Programs using programs, programs of type dynamic are ignored as current objects.
- Programs of this type can only have children of type program (via "Uses PR concept"). The linked programs document the possible implementations. Therefore they all must use the same programtype and member name.

See *Documenting Dynamic Structures*.

Metadata Diagram

The diagram below is an extract of the metadata structure in Predict showing the object type library structure and its associations.

A library structure can have system children of type Application Library.

The first system of type Application Library is the main library, the other child systems are the steplibs. The order of the children in the link list reflects the steplib hierarchy.
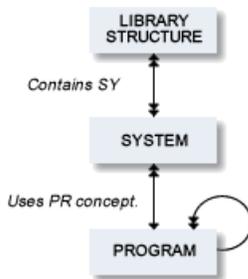
Active Retrieval Functions

The following active retrieval functions evaluate library structures to retrieve documentation and XRef data according to a specified steplib structure:

- **Programs using programs**
- **Systems containing programs**

Documenting Dynamic Structures

Example

In the example below, member MENU calls one of two INIT members depending which library is active at runtime.
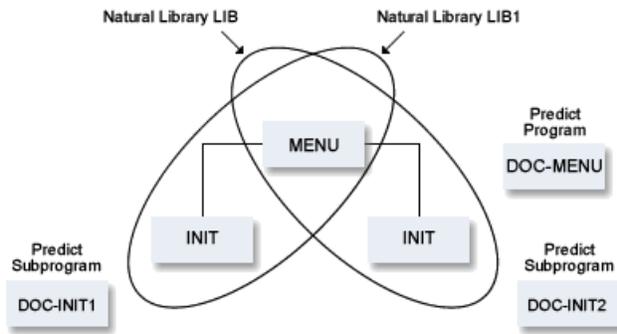
Documentation without Library Structure

The following table shows the objects needed to document the structure above without evaluating the steplib structure.

Object	Type	Subtype	Implementation Pointer			
			Member	Library	Fnr	DBnr
DOC-LIB-MAIN	System	Application Library	-	LIB-MAIN	54	180
DOC-LIB	System	Application Library	-	LIB	54	180
DOC-LIB1	System	Application Library	-	LIB1	64	180
DOC-MENU	Program	Program	MENU	LIB	54	180
DOC-INIT1	Program	Subprogram	INIT	LIB	54	180
DOC-INIT2	Program	Subprogram	INIT	LIB1	64	180

Enter DOC-INIT1 and DOC-INIT2 in the Program>Program link list of DOC-MENU. This has the disadvantage that you cannot tell which INIT member will be called by member MENU at runtime.

Documentation with Library Structure

The following table shows the objects needed to document the structure above in such a way that the steplib structure is evaluated.

Object	Type	Subtype	Implementation Pointer			
			Member	Library	Fnr	DBnr
DOC-LS1	Lib. Structure					
DOC-LS2	Lib. Structure					
DOC-LIB-MAIN	System	Application Library	-	LIB-MAIN	54	180
DOC-LIB	System	Application Library	-	LIB	54	180
DOC-LIB1	System	Application Library	-	LIB1	64	180
DOC-MENU	Program	Program	MENU	LIB	54	180

Object	Type	Subtype	Implementation Pointer			
			Member	Library	Fnr	DBnr
DOC-INIT	Program	Dynamic	INIT	See note below		
DOC-INIT1	Program	Subprogram	INIT	LIB	54	180
DOC-INIT2	Program	Subprogram	INIT	LIB1	64	180

Enter DOC-INIT (program of type Dynamic) in the link list of DOC-MENU. Member MENU will call up the member documented by DOC-INIT1 or DOC-INIT2 at runtime depending on the library structure.

Steplib Support with Active Retrieval Functions

The following active retrieval functions use library structures:

- **Programs using programs**
- **Systems containing programs**
- Programs using files

If the first two functions listed above are executed with the parameter Library structure, the steplib structure documented by the corresponding library structure object is evaluated.

> **Note:**   These functions are also described in the section *Active Retrieval* in the *Predict Reference* documentation. The descriptions there apply *without* evaluating the steplib structure.

Function Program using Program

This section describes the active retrieval function Program using program where a library structure is evaluated. See the section *Active Retrieval* in the *Predict Reference* documentation for a description of this function without evaluating the library structure.

Specifying the Library Structure

- Enter fully qualified library structure ID. Asterisk notation can be used to select one library structure from a list.
- If the implementation pointer of the main library in the library structure is incomplete, a window appears in which you must enter the missing parameters Library, Fnr, or DBnr.

```
                 +---------------- Additional criteria --------------+
   Retrieval t ! Main Library/first Natural Library of the           !
   Output mode ! Library Structure DOC-LS1                           !
   Program ID  ! has no qualified Implementation Pointer.            !
   in system . ! Please enter following parameters:                  !
   Member .... !                                                     !
   Library ... ! Library .......... ARH1                             !
   Library str ! Fnr ..............  64                              !
   Entry ..... ! DBnr ............. 180                              !
   Restriction !                                                     !
   Output opti ! Press ENTER to confirm                              !
               +-----------------------------------------------------
```

If the main library in the library structure is a 3GL library, the first Natural library is taken as main library and must be given a fully qualified implementation pointer if required.

▪ The implementation pointer defined for the main library (or first Natural library) is used to append all other incomplete implementation pointers for all other libraries in the library structure.

▪ If you set the output option Cover page to Y, the library structure with complete implementation pointers for all libraries is displayed:

```
                   -------------- Cover page ---------------
   Program ID ... DOC*

   Library structure ID .. DOC-LS1

                          Library    Fnr   DBnr
                          ARH1        64    180
                          ARH3        64    180
                          ARH         54    180
                          ARH3GL2    255    255
```

In the example above, the missing values have been appended with the DBnr 180 specified under Additional criteria above.

Determining the Current Objects to be Output

The following rules apply:

▪ Programs of type Dynamic are ignored as current objects.

▪ Only programs with identical implementation information to a system contained in the library structure are output:

- if the implementation pointer of the program object is complete (Member, Library, Fnr, DBnr), this is evaluated;

- if the implementation pointer of the program object is incomplete, the XRef data is evaluated.

Determining the Related Objects to be Output

Related objects are evaluated against documentation data (implementation pointer) and XRef data. The following rules apply:

- If a program of type Dynamic is linked to the current object via "Uses PR concept", this link must be resolved. The programs represented by the program of type Dynamic are checked, and the program with the implementation pointer that best matches the library structure replaces the program of type Dynamic.

- During checks as to whether a program is implemented, the entire library structure is evaluated. If the implemented member is found in a steplib, the program is marked as I and the comment >>>impl. in steplib XXXXXXXX DBnr 99999 Fnr 99999<<<.

Sample Output

The screen below shows sample output for function Programs using Programs.

```
  13:27:36              *****  P R E D I C T  *****              2007-05-31
                        - List Program Using Programs -

  Program ID ......... * DOC-MENU
  Type ............... Program
  ----------------------------------------------------------------------------

  Implementation
    Member .. MENU    Library .. ARH     Fnr .. 54  DBnr .. 180

  Cnt  Program ID                      Ty La Member   Library  Fnr DBnr L D I U

     1 DOC-INIT2                       P  N  INIT     ARH1      64  180 L D
       >>> Dynamic call defined in  DOC-INIT  <<<


  *** End of report ***




  Command ===>                                             Scroll ==> CSR
  Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
                  Quit        RFind Flip  -     +         Left  Right
```

Comments

Program DOC-INIT2 is linked via "Uses PR concept" to a program of type Dynamic named DOC-INIT. The link list of the dynamic program DOC-INIT is checked and the program with the most complete implementation pointer is given.

Function Systems containing Programs

This section describes the active retrieval function Systems containing programs where a library structure is evaluated. See the section *Active Retrieval* in the *Predict Reference* documentation for a description of this function without evaluating the library structure.

Specifying the Library Structure

■ Enter fully qualified library structure ID. Asterisk notation can be used to select one library structure from a list.

■ If the implementation pointer of the main library in the library structure is incomplete, a window appears in which you must enter the missing parameters Library, Fnr, or DBnr.

```
                +---------------- Additional criteria --------------+
    Retrieval t ! Main Library/first Natural Library of the         !
    Output mode ! Library Structure DOC-LS1                         !
    Program ID  ! has no qualified Implementation Pointer.          !
    in system . ! Please enter following parameters:                !
    Member .... !                                                   !
    Library ... ! Library .......... ARH1                           !
    Library str ! Fnr ..............   64                            !
    Entry ..... ! DBnr ............. 180                             !
    Restriction !                                                   !
    Output opti ! Press ENTER to confirm                            !
                +---------------------------------------------------
```

If the main library in the library structure is a 3GL library, the first Natural library is taken as main library and must be given a fully qualified implementation pointer if required.

■ The implementation pointer defined for the main library (or first Natural library) is used to append all other incomplete implementation pointers for all other libraries in the library structure.

■ If you set the output option Cover page to Y, the library structure with complete implementation pointers for all libraries is displayed:

```
                    -------------- Cover page ---------------
    System ID ... DOC*

    Library structure ID .. DOC-LS1

                       Library   Fnr   DBnr
                       ARH1       64    180
                       ARH3       64    180
                       ARH        54    180
                       ARH3GL2   255    255
```

In the example above, the missing values have been appended with the DBnr 180 specified under Additional criteria above.

Determining the Current Objects for Output

XRef data is evaluated, and only systems with an identical implementation pointer to one of the systems in the library structure are given.

Determining the Related Objects for Output

A program is marked as implemented if it is contained in the current library. The current library is the library documented by the system specified.

A program is also marked as implemented if it is linked via "Uses PR concept" to a system documented as a steplib. (The program would not be marked as implemented if you were working without a library structure.)

A note is given indicating in which other steplib(s) of the library structure the program is implemented.

Steplib Support with LIST XREF for Natural

The three possible methods of evaluating XRef data are listed below. Select the method you require in the LIST XREF menu before you call a function. This option is valid for the duration of your session or until you select another option in the LIST XREF menu.

See also section *LIST XREF for Natural* in the *Predict Reference* documentation.

Using the command INFO you can display at any time during your LIST XREF session all libraries that will be evaluated by LIST XREF functions. The current library is marked with an arrow.

The Library Structure Documented in Predict

The link list Library structure to System is evaluated via "Contains SY". Each system in the list is checked as follows:

- If no information is present in the implementation pointer of the system, the system is ignored.

- If the implementation pointer is incomplete, the system searches for possible XRef data. This XRef data is used to supply the missing Fnr and DBnr information in the implementation pointer.

- If no XRef data is found, the values of the current FUSER file are used to supply the missing DBnr and Fnr information in the implementation pointer.

- If the current library is a Natural library, the structure is appended with --> *STEPLIB <--.

Runtime Structure

The runtime structure is determined as follows. The following rules apply:

- The current library always appears first in the list. If this library is documented in Predict, the corresponding system ID is also displayed.

- With Natural Security, up to 8 Libraries can be specified as steplibs with Library, DBnr and Fnr.

- The default steplib is declared in the Natural parameter moduleNATPARM parameter file or allocated with the dynamic parameter STEPLIB when starting Natural (*STEPLIB).

Without any Structure

The LIST XREF functions evaluate XRef data without specification of steplibs. Only objects in the current library are displayed.

Steplib Support in Batch Mode

In batch mode, too, there are three possible methods of evaluating XRef data:

- `STRUCTURE <structure-name>`
  With this command you can specify which library structure is to be used for evaluating XRef data.

- `STRUCTURE *R`
  This command specifies that the runtime structure is to be used for evaluating XRef data.

- **No Structure specified**
  If you do not specify any structure, LIST XREF functions work without evaluation of steplib specification.

Effects of Steplib Support on LIST XREF

Steplib support affects LIST XREF functions as follows. A distinction is made between Top-down and Bottom-up functions:

Top-down

Example: Function Program using program

```
09:50:43            *****  P R E D I C T  *****              2007-05-31
Library: PDLX              - Invoked Programs -        DBnr:  180 Fnr:   54
Command: PROG * (*) USING PROG * (*) WITH * VIA *                   Page:   1


     T:Program                 using                      via
------------------------------------------------------------------------------
   1 P:ZPDFIELD            1 N:N-BUFEDT <<- nfnd           Callnat

   2 P:ZPDP0              1 P:ZPDP1                        Fetch
                            via ZPDP&
                          2 S:ZPDS1  (NEWDICLX,180,54)     Perform
                            Function: SUB-IN-ZPDS
```

Comments

Only programs in the current library that call other programs are displayed.

to 1	The note <<- nfnd means that the called program was not found within the structure specified. "Not found" in this context means that no XRef data is present.
to 2	If the called program is not contained in the current library, it is displayed in parentheses with DBnr andFnr.

Bottom-up

Example: Function Programs referenced in programs

```
    09:53:09              *****  P R E D I C T  *****                2007-05-31
    Library: PDLX                - Invoked Programs -      DBnr:   180 Fnr:    54
    Command: PROG * (*) REF PROG * (*) WITH * VIA *                   Page:    1

         T:Program                   referenced in              via
    ------------------------------------------------------------------------------
      1 ?:*DYNAMIC
                               1 P:ZPDP3                       Fetch
      2 ?:N-BUFEDT
                               1 P:ZPDFIELD                    Callnat
      3 M:ZPDM1
                               1 P:ZPDP1                       Map
                               2 P:ZPDP2  (NEWDIC,180,54)      Map
      4 P:PGMCO002 (*SYSCOB*,255,255)
         Entry  : PGMCO002
                               1 P:ZPDP1                       Call
                               2 P:ZPDP2                       Call
                               3 S:ZPDS1                       Call
```

Comments

to 1	*DYNAMIC produces a list of all programs that call up other programs by means of variables:
	
	`ASSIGN #A = 'SUB1'`
	`FETCH #A`

to 2	The question mark means that the program N-BUFEDT was not found within the specified structure. "Not found" in this context means that no XRef data was found for the program object. The program is, however, referenced by program P:ZPDFIELD via CALLNAT.
to 3	Member ZPDM1 was found within the current library. If the called program is contained in the current library (here ZPDM1), programs not contained in the current library that call this program are also displayed (here ZPDP2 in Library NEWDIC). Library, DBnr, Fnr are displayed in parentheses.
to 4	The called program (PGMCO002) was found, but in another library within the structure (library *SYSCOB*). In this case only calling programs within the current library are displayed.

Example: Function Program referenced in programs recursively

```
13:44:35              *****  P R E D I C T  *****                2007-05-31
Library: NEWDICLX           - Invoked Programs -        DBnr:  180 Fnr:    54
Command: PROG XHMENU10 (*) REF REC * (*) WITH * VIA *            Page:    1
         DEPTH 7
     1 M:XHMENU10
1 ---------2 ---------3 ---------4 ---------5 ---------6 ---------7 ---------
P:XPHELP   P:XPCIMPL  P:XPVERI   M:XMCIMP00 P:XPCIMPL  <--- rec
                                 M:XMREFE00 P:XPREFE   P:XPVERI   <--- rec
                                 P:XPCIMPL  <--- rec
                                 P:XPCMDP   M:XMCIMP00 <--- suppr
                                            M:XMCONS00 P:XPVCONS  P:XPVERI
                                                                  <--- rec
                                            M:XMCOPY00 P:XPCOPY   P:XPMENU
                                                                  <--- steplib




             ↵
```

Comments

The note <--steplib means that the evaluation was stopped at this point because the called program is contained in another library.

```
09:44:21              *****  P R E D I C T  *****                2007-05-31
Library: NEWDICLX           -   XRef Menu   -          DBnr:  180 Fnr:    54


Structure ...........: LS-NEWDICLX


          System Id                          Library  Fnr DBnr
          ------------------------------     -------- --- ----
          PD-COB                             *SYSCOB* 255  255
        -->PD-NEWDICLX                       NEWDICLX 180   54
          --> *STEPLIB <--                   SYSTEM   180   54
```

4 Adabas Vista

Storing data in individual files of databases on local machines that are not integrated in any network is a rather limited approach when designing large and complex applications. To gain flexibility and safety, data can be distributed across several Adabas databases which may reside on different machines. Such distributed data structures can be realized with the Software AG product Adabas Vista.

Distributed data structures for use with Adabas Vista and can be defined in Predict, and the objects necessary to implement the structures physically can be generated from these definitions.

It is important to understand that storing data in the good old-fashioned way (simple files residing in isolated databases on local machines) also establishes a data distribution structure, albeit a very simple one. The description given below therefore also applies to the definition of simple files.

> **Note:** This section applies exclusively to Adabas files. See the Adabas Vista documentation for a complete description of this product.

The Software AG product Entire Transaction Propagator can also be used to define distributed data structures.

Different Types of Data Distribution

Adabas Vista offers various options for distributing data across a network.

- **Storing Data Locally in an Isolated Database**
  Adabas Vista is not required if data is stored locally in a single database. In this case, the database is called an isolated database and the logical distribution type of all files is either simple or expanded.

- **Distributing/Duplicating Data Across Several Databases (Adabas Vista)**
  With Adabas Vista, data belonging to one logical file can be physically distributed across several physical files (that may reside in different databases).

  Data logically belonging to one file can be split between several physical files. For example: A (logical) file is defined to store information on all customers of a company. Data of customers living in the north is to be stored separately from the data of customers living in the south of the country. The zip code is used as the distribution criterion.

  A file with the logical Distribution type partitioned can be used to store data in this fashion.

- **Storing Data in Remote Databases (Net-work)**
  With Net-work, data stored in databases on remote systems can be accessed as if it were stored locally. The use of this product is described in the documentation of Net-work.

■ **Distributing/Duplicating Data Across Several Databases on Different Machines**
By combining Net-work and Adabas Vista, data belonging to one file can be distributed across
several databases residing on different machines.

Defining the Distribution of Data in Predict

General Information

Predict objects of type network, virtual machine, database and file are used to define where exactly
data is stored.

Adabas attributes of Predict file objects define how files are implemented in a database, for example
partitioned. Adabas attributes document the physical links between files and databases.

Vista elements document the accessibility of the physical files with Adabas Vista.
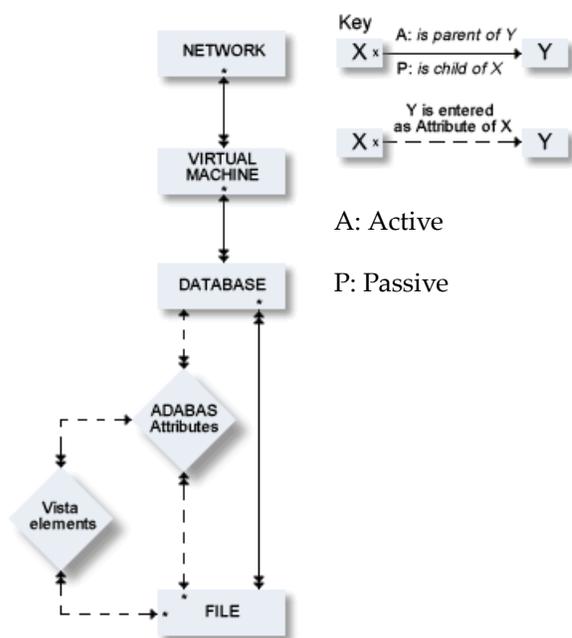
Links between networks, virtual machines, databases and files are defined as follows:

■ Links between networks, virtual machines, databases are defined with attributes of the respective
lower level objects, for example: the link between a network and a virtual machine is defined
with the parameter in Network of the virtual machine.

Each virtual machine object must be linked to a network object, and each database object must
be linked to a virtual machine object.

- Links between databases and files are defined with the function Link children of association "Contains FI".

- Information on how files are implemented in a database is stored in the Adabas attributes of file objects.

  Adabas attributes can be modified by entering the line command `.A` when editing the file list of a database or with the file maintenance function Modify Adabas attributes.

Defining the Distribution of Data

Defining the distribution of data is a two-step process:

1. Define the structure of the data distribution by creating and linking the respective network, virtual machine, database and file objects. See the sections *Defining a Network, Virtual Machine and Database Structure* and *Defining the File Structure*.

2. Determine the accessibility of data by creating Vista elements for physical file definitions. A Vista translation table can be generated from Vista elements. See the section *Vista Translation Table* in the section *Generation* in the *External Objects in Predict* documentation.

Defining a Network, Virtual Machine and Database Structure

Since data can be distributed across several databases, the exact location of data storage has to be specified: each database object must be linked to a virtual machine and each virtual machine must be assigned to one network.

Defining Networks and Virtual Machines

Networks and virtual machines identify the location of databases.

What is a Network?

- A network contains all virtual machines and databases that are to be accessed. In the case of databases that reside on local machines without any remote databases being connected, a network may in fact identify a local machine.

- A network object HOME is provided by Predict.

What is a Virtual Machine?

- A Predict object Virtual Machine identifies a machine and operating system environment of databases. A virtual machine represents one Adabas SVC (supervisor call).

- Each virtual machine can contain one Transaction Manager used to distribute Adabas calls across the network. However, a virtual machine does not necessarily have to contain a Transaction Manager.

- Each virtual machine can contain one or more Vista databases providing access to partitioned data. However, a virtual machine does not necessarily have to contain a Vista database.

Attributes of Networks and Virtual Machines

- A network object has all the standard attributes of Predict objects (for example extended description and abstract) and no type-specific attributes.

- A virtual machine object has all the standard attributes plus the type-specific attribute Operating system type.

> **Note:** Network and virtual machine attributes are described in detail in the respective sections of the *Predefined Object Types in Predict* documentation.

Defining a Database

```
   13:36:38              *****  P R E D I C T  *****              2007-05-31
                            - Add a database -
   Database ID ..... DATABASE-TEST          +All--------Run mode----------+
                                            ! _ I    Isolated            !
                                            ! _ L    Local               !
                                            ! _ V    Vista               !
         Database type .............* A Adabas   ! _                     !
         Belongs to VM .............* HOME       ! _                     !
         Run mode ..................* * Vista    ! _                     !
         Physical database number ..*           ! _                     !
                                            ! _                          !
```

> **Note:** Database attributes are described in detail in the section *Database* in the *Predefined Object Types in Predict* documentation.

Parameters	
Belongs to VM	Associates the database to a virtual machine. Must be specified for all databases except types Conceptual, DB2 and IMS. A default virtual machine can be defined in the profile. See *Maintenance Options* in the section *Predict User Interface* in the *Introduction to Predict*.
Run mode	Determines the use of the database with respect to the distribution of data with Adabas Vista. Valid Values: Isolated, Local and Vista. **Note:** The Predict parameter Run mode corresponds to the ADARUN parameters Vista and Local. The corresponding values in Predict and ADARUN are shown in the table below:

Parameters			

Predict	ADARUN		
Run mode	Vista	Local	
Isolated	-	-	
Local	-	Yes	
Vista	Yes	-	

I	Isolated. Vista is not used. The database may be accessible using Net-work. A database of type isolated can only contain files of the types simple and expanded.
L	Local. The database cannot be accessed using Net-work.
V	Vista. Adabas Vista is used.

Physical database number	Identifies a database in a virtual machine and, in the case of databases that can be accessed using Adabas Vista, in a network.

Physical database number — Identifies a database in a virtual machine and, in the case of databases that can be accessed using Adabas Vista, in a network.

- If a database can be accessed using Vista (Run mode V is set to Y or N), the Physical database number must be unique throughout the network.

- If a database is of type isolated or local, the Physical database number must be unique only within a virtual machine.

  However, it is recommended to use physical database numbers that are unique throughout a network for local and isolated databases as well.

The uniqueness of physical database numbers can be forced by setting the parameter Unique-DBnr/Fnr in the second Miscellaneous screen of the General Defaults function. See the section *Defaults* in the *Predict Administration* documentation.

Defining the File Structure

Physical File Level

Physical file:
DBnr: 3
Fnr: 13
Physical distribution
type: Partitioned

Database 3

Logical File Level

Physical file:
DBnr: 2
Fnr: 12
Physical distribution
type: Partitioned

Database 2

Logical file:
File ID: Test-file
File type: Adabas
Logical distribution
type: Partitioned
Logical Fnr: 11

Physical file:
DBnr: 1
Fnr: 11
Physical distribution
type: Simple

Database 1
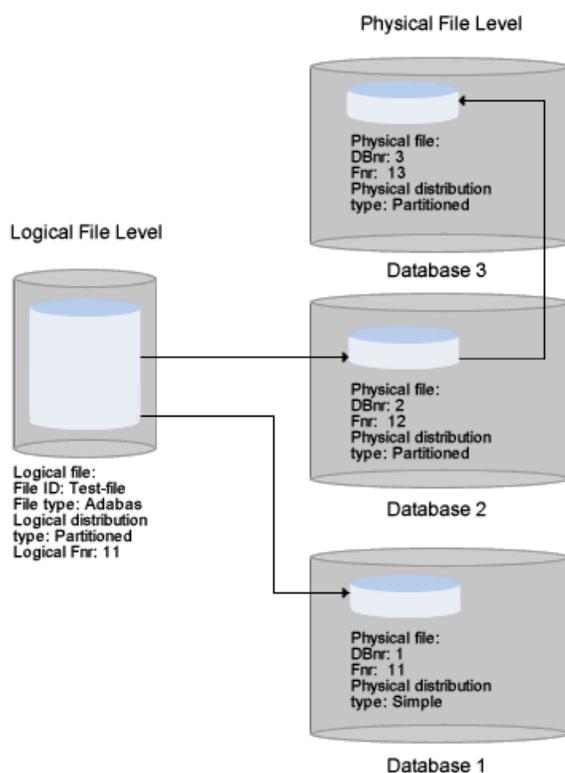
Description of the Structure

The above diagram shows how data distribution is defined on two levels:

- **The logical level**
  The file *Test-file* has the logical distribution type partitioned.

- **The physical level**
  The data is distributed across the physical files 12 and 13 in databases 2 and 3. It also exists as physical file 11 of type simple in database 1 (for example for evaluation purposes).

Defining a File Structure Logically and Physically

The diagram shows that files are defined on the logical and the physical file level:

■ **On the logical level**
Defining a logical file includes the definition of the fields in a file. Files of type simple, expanded, partitioned and PROPAGATOR can be defined on the logical level. See *Defining a Logical File*.

■ **On the physical level**
The exact physical implementation for the storage of data is defined. Depending on the logical distribution type, different types of physical files can be implemented. The following table shows which types of physical files can be used for files with different logical Vista types. See *Defining the Physical Implementation of Logical Files*.

Logical Level	Physical Level
simple	simple
expanded	expanded, simple
partitioned	partitioned, simple
PROPAGATOR	PROPAGATOR master, PROPAGATOR replicated, simple

Defining a Logical File

In the first screen that is displayed when you define a file, basic attributes of the file are specified. The parameters that are important when defining files for use with Adabas Vista are described below. See also section *File* in the *Predefined Object Types in Predict* documentation.

```
  13:01:26                ***** P R E D I C T *****                  2007-05-31
                               - Add a file -
  File ID ......... ADABADA-33                    +All---Distribution types-----+
                                                  ! _ E   Expanded file       ↵
  !
                                                  ! _ P   Partitioned         ↵
  !
                                                  ! _ N   PROPAGATOR file      ↵
  !
                                                  ! _ ' ' Simple file          ↵
  !
                                                  ! _                          ↵
  !
                                                  ! _                          ↵
  !
              File type ................* A Adab ! _                          ↵
  !
              Master file ..............*        ! _                          ↵
  !
              File number ..............* 123    ! _                          ↵
  !
              Logical distribution type .* * Simp ! _                         ↵
  !
              Contained in DA ...........*        ! _                          ↵
  !
```

Parameters		
File number	If a database is specified, the file number of the logical file is taken as a physical file number automatically (if this is possible). If not, a free physical number can be selected from a selection window. The file number must be in the range 1 - 32767.	
Logical distribution type	Determines how the file can be implemented in an Vista or Entire Transaction Propagator environment.	
	E	Expanded: to be implemented in several physical files with identical FDT but different data in each physical file (continuous file). All files are located in the same database.
	P	Partitioned: as expanded but data is accessed with Adabas Vista and can be distributed over several databases.
	N	Propagator file: to support multiple, partially replicated copies of one physical file with Entire Transaction Propagator.
	blank	Simple file (default).
Contained in DA	A link to this database is established.	

Defining the Physical Implementation of Logical Files

A logical file definition does not contain any information on the physical implementation of the file. To specify the physical implementation of a file, a physical file definition has to be added. Physical file definitions are identified by both a physical database number (PDBnr) and a physical file number (PFnr).

The physical database number and a physical file number are specified in one of the following ways:

■ A database is specified by adding the logical file (with the in database parameter). The PDBnr is taken from the Predict database object. The PFnr is the same as the logical file number (the File number) if this is possible. If not, a free physical number can be selected from a selection window.

■ By specifying a database ID when adding new physical file in the Select one or more physical files window as described in the next section.

■ By executing the .A line command in the list editor when maintaining the file list of a database object. If a file is not yet in the database, a new physical file is added.

> **Note:** If no physical file has yet been defined for a logical file, the string ** default record ** is displayed in the Select one or more physical files window.

Adding, Modifying and Purging Physical Files

Physical file definitions are added or modified using the Select one or more physical files window. This window appears when either

■ the Modify Adabas Attributes Function (Code J) in the File Maintenance menu is executed or

■ the Additional attributes parameter in the first Add/Modify file screen is set to Y.

The Select one or more physical files window contains a list of all physical files belonging to the logical file.

Physical file definitions are added, modified or purged with one-letter commands in the column Cmd.

> **Note:** The Select one or more physical files window is also displayed if the logical file contains only one physical file (as shown in the screen below).

```
   08:57:25              *****  P R E D I C T  *****              2007-05-31
                             - Add a file -
   File ID ......... JPE-PART5                    +--- Additional attributes ---+
   Type ............ Adabas, Partitioned          ! --> Mark one or more        !
   File number ..... 923   +All--------Select one or more physical file-------+
   Contained in DA . HNO-D ! Cmd Database name                    T PDBnr PFnr !
   Keys ..                 !                                                   !
                           ! _    ** new **                                    !
   Literal name .......... ! _  HNO-DA-A                          P 134   125  !
   Average count ......... !                                                   !
   Stability ............. !                                                   !
   Sequence field ........ !                                                   !
   Vista Access DBnr ..... !                                                   !
   Vista Access Fnr ...... !                                                   !
   Adabas SQL usage ...... !                                                   !
   Abstract    Zoom: N     !                                                   !
                           !                                                   !
                           +---------------------------------------------------+
                                      !                                        !
   EDIT:   Owner: N   Desc: N         Has Field+----------------------------+
```

Commands in the Select one or more physical files window	
A	Add a new physical file definition. A can only be entered in the line **new** at the top of the list. The Add command displays a window to enter a physical database ID and subsequently the Modify Adabas Attributes screen.
M, X or /	Modify the physical file definition. The Modify Adabas Attributes screen is displayed.
P	Purge the physical file definition. Additional confirmation is requested. The physical file is removed from the file list of the database.

Specifying the Vista Attributes of Physical Files

```
    13:47:38            *****  P R E D I C T  *****           2007-05-31
                          - Add Adabas attributes -
    File ID ......... JPE-PART4                  +-----Additional attributes-----+
    Type ............ Adabas, Partitioned        ! --> Mark one or more           !
    Contained in DA . HEB-DA-3 (PDBnr: 33333)    !      attributes                !
                                                 !   _  Phys. distribution attr.  !
    Required attributes                     Ph   !   _  Miscellaneous attributes  !
      Phys. file number ..* 146                  !   _  ADAM key definition       !
      Min ISN ............. 1                    !   _  Extent allocation         !
      Max ISN .............                      !   _  Distribution criteria     !
                                                 !   _  Encodings                 !
          Device      Cylinder Blocks   Paddin   !                                !
          *-----      -------- --------  ------   !                                !
    ASSO    3380   UI                            !                                !
                   NI                            !                                !
    DATA    3380   DS                            !                                !
                                                 !                                !
    Loading attributes                      Lo   !                                !
     Max recl. ...........                       !                                !
       ISN reusage ......... N (Y,N)             !                                !
       User ISN ........... N (Y,N)              +--------------------------------+

       Additional attributes ..* N        Associations ..* N
```

The Additional attributes window that is displayed by entering Y in the Additional attributes field of the Modify Adabas Attributes screen (see screen above) contains two topics needed for defining data distribution:

- Physical distribution attributes

- Distribution criteria

Both topics are described in the sections *Specifying Physical Distribution Attributes* and *Specifying Distribution Criteria for Partitioned Files* below. All general attributes of physical file definitions are described in the section *Adding, Modifying and Purging Physical Files*.

Specifying Physical Distribution Attributes

To specify or modify the Vista attributes of a physical file, select the topic Phys. distribution attr. in the Additional Attributes window. Physical distribution attributes is not contained in the Additional attributes window if no association to a database exists or the logical distribution type is simple.

```
   13:58:35              *****  P R E D I C T  *****              2007-05-31
                         - Modify Adabas attributes -
   File ID ......... PD-A-EXP                        Modified 2007-05-31 at 13:24
   Type ............ Adabas, Expanded file                     by PD
   Contained in DA . PD-AAA (PDBnr: 28)


   Distribution attribute
     Phys. distribution attr. ......* E   Expanded file



   Loading attributes
     Min ISN .................. 1
     Max ISN ..................
     One AC extent ............ Y (Y,N)
```

Parameters	
Type	The distribution types to be assigned to a physical file. The table below shows which types of physical distribution types apply to different logical distribution types: **see table below**
Loading attributes	
Min ISN	ADALOD LOAD parameter MINISN
Max ISN	ADALOD LOAD parameter MAXISN
One AC extent	ADALOD LOAD parameter NO AC EXTENSION.

Logical distribution type	Physical distribution type
simple	simple
expanded	expanded, simple
partitioned	partitioned, simple
PROPAGATOR	PROPAGATOR master, PROPAGATOR replicated, simple

Specifying Distribution Criteria for Partitioned Files

The distribution criteria are used as follows:

For files with logical distribution type partitioned, the distribution criteria determine how data is split across several physical files.

Any field of a file can be taken as the distribution criterion. An example: The field zip_code is evaluated. Only if a record has a zip_code starting with 6 (identifying the area around Frankfurt/Main) but equal or less than 61999 is a record to be included into the file. The respective input is shown in the diagram below.

```
   08:13:12              *****  P R E D I C T  *****              2007-05-31
                          - Modify Adabas attributes -
   File ID ......... PD-A-PAR                         Modified 2007-05-31 at 08:12
   Contained in DA . PD-A0                                       by SMR
   PDBnr ........... 16    PFnr ... 151


   Ty Partitioning field              F  Cs Length   Occ   D U DB N NAT-l
   -- *------------------------------ -- -- -------- ----- - - -- - -----
      ZIP-CODE                        N  5.00                 AH N

      1 Access ....* GE  Critical .. (Y,N)    Shared Partition .. (Y,N)
        Part. name . Frankfurt
        High value . 61999                                            Zoom: N

      1 Access ....* GE  Critical .. (Y,N)    Shared Partition .. (Y,N)
        Part. name . Munich
        High value . 82999                                            Zoom: N

      1 Access ....* GE  Critical .. (Y,N)    Shared Partition .. (Y,N)
        Part. name . Hamburg
        High value . 22999                                            Zoom: N


      Additional attributes ..* N           Associations ..* N        Scroll to:
```

For an explanation of the valid parameters and values see *Specifying Restrictions on Input Data - Distribution Criteria* in the section *Adabas Files, File Type A* in the *Predefined Object Types* documentation.

Including the Definition in the Vista Table

To access data in physical files with Adabas Vista, the file definitions must be contained in the Vista translation table of the Adabas Vista translator database. Exactly one Vista translator database must exist in any Virtual Machine (see also description of the Vista parameter in the section *Defining a Database*).

Vista translation tables can be generated from Vista elements defined in Predict. See *Vista Translation Table* in the section *Generation* in the *External Objects in Predict* documentation.

Vista elements on file level are defined with the file maintenance functions Add/Modify Vista elements.

Vista elements on database level are defined with the database maintenance functions Add/Modify Vista elements.

The function uses the following screen:

```
 13:30:02                 *****  P R E D I C T  *****              2007-05-31
                             - Add Vista element -
 File ID ......... HNO-FI-V                          Added 2007-05-31 at 13:30
 Type ........... Adabas, Partitioned                      by HNO


 Network ........* HOME
 Simple .......... Y (Y,N)              Partition ID assignment ..* V Vista
 Vista                                  Max number of partitions .. 255
  Environment ID .                      Enable Read-by-ISN ........ Y (Y,N)
  DBnr ...........                       Part. file concurrency .... 8
  Fnr ............                       Store control option .....* 1 Reject
  Name .......... HNO-FI-V

     Database                         PDBnr PFnr  Criterion
     *--------------------------- ----- ----- -------------------------------
   1



 Additional attributes ..* N           Associations ..* N              Scroll to:
```

Parameters	
Type	Type of the logical file (for example Adabas, Partitioned). A read-only field.
	Note: The subsequent parameters Network, Environment ID and Simple can be specified when adding a Vista element. When modifying a Vista element, these fields are read-only.
Network	The Vista element is available throughout the given network.
Simple	Y	Only physical files of type simple can be accessed with this Vista element.
	N	Physical files of all other suitable types can be accessed with this Vista element. Which types are suitable depends on the logical Vista type of the file. See *Specifying Vista Attributes*.
Environment ID	The Vista element can be used exclusively by the given environment. If, for example, a data administrator wants to access a file for administration purposes, he might create a Vista element for his privileged use.
DBnr	Database number used for access from the application. This number is translated into the PDBnr by Vista.
Fnr	File number used for access from the application. This number is translated into the PFnr by Vista.
Name	Name of the translation element in Vista.
Database	Database containing the physical file.

Parameters	
PDBnr	Physical database number.
PFnr	Physical file number.
Criterion	Name of the distribution criterion.

Vista Key

The Vista element attributes Network, Environment ID and Vista numbers together identify how a Vista element can be used. These attributes are also referred to as the Vista Key. The following rules apply:

▪ Environment ID and Vista numbers must be unique for each Vista element within each network.

▪ If the parameter Unique DBnr/Fnr in the Predict defaults is set to Y, Environment ID and Vista number must additionally be unique throughout all networks.

Retrieving Information on the Use of Vista Numbers

The function Vista number (code N) in the Network Retrieval Menu can be used to determine how Vista numbers are referenced in databases, physical files and Vista elements. See the section *Network* in the *Predefined Object Types in Predict* documentation.

Generating, Incorporating, Comparing and Maintaining Data Definitions under Adabas Vista

Predict generation, incorporation and comparison functions can be applied to data definitions under Adabas Vista. The following functions are designed especially for maintaining Vista translation tables:

▪ Generation of Vista translation tables (command: `GENERATE STARTAB`)

▪ Incorporation of entries in Vista translation tables as Vista elements of Predict database and file objects of type A with the function Incorporate Adabas Database/File. Incorporating Vista elements requires that either a Predict file object for the implemented physical file does not exist, or an existing Predict file object has the correct physical distribution type.

▪ Comparison of Vista translation tables with Adabas file definitions in Predict (command: `COMPARE VISTA-FI`).

Using the above functions requires the following:

▪ An interface that is provided with Adabas Vista Version 7.4. If you want to use this interface together with Adabas Vista Version 7.3, please contact Software AG.

■ LFILE 152 must be set and must point to the Vista system file.

For detailed descriptions of the above options see the respective parts of the sections *Generation*, *Incorporation* and *Comparison* in the *External Objects in Predict* documentation.

5 VSAM

Documenting VSAM

VSAM files and userviews can be documented in Predict with four different types of Predict file objects.

- **File type V**
  Physical VSAM file (master file)

- **File type L**
  Logical VSAM file (master file). This type can only be applied to VSAM files using KSDS (key-sequenced data set).

- **File type W**
  Userview of physical VSAM file

- **File type R**
  Userview of logical VSAM file. This type can only be applied to VSAM files using KSDS (key-sequenced data set).

The different file types are described in detail below.

Physical VSAM file - Master File, File Type V

File type V is used for the documentation of a physical VSAM file in Predict.

Field definitions of a physical VSAM file have the same structure as definitions of a sequential file: the position of a field cannot be specified directly but is determined by its offset. The offset is calculated from the lengths of the fields already defined. Therefore DUMMY fields must be defined if space is to be left free between two fields (see examples below).

Example

```
File ID..: EXAMPLE-V

File-Type: V

DD name..: EXAMDD


L  Field-name     F Length  D    Offset    Remark

----------------------------------------------------------

1  DUMMY1         A  10.0           0

1  PRIM-KEY       A  15.0     P    10       Primary key
```

```
1   DUMMY2          A   8.0           25

1   ALT-KEY1        A   5.0    A      33      Alternate Key

1   ALT-KEY2        B   7.0    A      38      Alternate Key
```

In this physical file definition, only the keys of the VSAM file (DD name: EXAMDD) are defined. This physical VSAM file is to be used in connection with the two logical VSAM files (*EXAMPLE-L1* and *EXAMPLE-L2*) which are shown below.

The primary key field has the length 15.0. In the definition of logical VSAM files *EXAMPLE-L1* and *EXAMPLE-L2* below, these 15 places are used for the storage of the record type specifying VSAM prefix (length 9.0) and the primary key of the logical file (length 6.0). The DUMMY field (A 10.0) in the beginning ensures that the primary key field position matches the field definitions for the VSAM prefix and primary key in the logical files.

Logical VSAM File - Master File, File type L

Logical VSAM files can be documented with file type L. A logical VSAM file defines a record layout for use in a physical VSAM file. By using logical VSAM files information objects of different types (and correspondingly different record layouts) can be stored in one physical VSAM file. See also *Using a Record Layout Concept*.

The following rules apply when defining a logical VSAM file:

- Before a logical VSAM file can be documented in Predict, the physical VSAM file to which the logical file belongs must have been documented.

- Position and length of fields in logical VSAM files are defined in the same way as in physical VSAM files.

- Records in a VSAM data set belonging to a logical file are identified by a VSAM prefix. For use with Natural for VSAM the field for the VSAM prefix has to start at the same position as the primary key in the physical VSAM file.

- Therefore the length of the field for the primary key in a logical file can be calculated as follows (see also examples below):

```
length of primary key in logical file =
length of primary key in physical file - length of VSAM prefix
```

- The value of the prefix can be specified explicitly for each logical file. If the VSAM prefix is to be specified with trailing blanks, each blank must be replaced with a special VSAM trailing blank character. This special character is defined with the Modify DDM defaults function.

- If no VSAM prefix is specified explicitly, the rightmost three digits of the file number are used as the VSAM prefix. The field defined for the prefix then has to have the length 3.0.

■ Alternate keys must be defined with the same offset and length in a logical and the corresponding physical VSAM file.

Examples

```
File ID.....: EXAMPLE-L1

File-Type...: L

Related file: EXAMPLE-V

VSAM prefix.: RECTYPE-A


L  Field-name    F Length  D  Offset   Remark

------------------------------------------------------

1  FIELD-A-1     A  2.0         0

1  FIELD-A-2     A  8.0          2

1  VSAM-PREFIX   A  9.0         10      VSAM prefix

1  PRIMKEY-A     A  6.0    P    19      Primary key

1  FIELD-A-3     A  1.0         25

1  FIELD-A-4     A  2.0         26

1  FIELD-A-5     A  5.0         28

1  ALT-KEY1      A  5.0    A    33      Alternate Key

1  ALT-KEY2      B  7.0    A    38      Alternate Key

1  FIELD-A-6     P  2.5         45

1  FIELD-A-7     N  8.2         49
```

```
File ID.....: EXAMPLE-L2

File-Type...: L

Related file: EXAMPLE-V

VSAM prefix.: RECTYPE-B


L  Field-name     F Length  D   Offset   Remark

------------------------------------------------------

1  FIELD-B-1      B  6.0          0

1  FIELD-B-2      A  4.0          2

1  VSAM-PREFIX    A  9.0          10      VSAM prefix

1  PRIMKEY-B      A  6.0     P    19      Primary key

1  FIELD-B-3      A  3.0          25

1  FIELD-B-5      A  5.0          28

1  ALT-KEY1       A  5.0     A    33      Alternate Key

1  ALT-KEY2       B  7.0     A    38      Alternate Key

1  FIELD-B-6      N  5.3          45

1  FIELD-B-7      B  18.0         53
```

The fields for storage of the VSAM prefix identifying the record type starts in the same position as the primary key in the corresponding physical VSAM file *EXAMPLE-V* above. The length of the fields VSAM-PREFIX and PRIMKEY-A (or PRIMKEY-B) together is 15.0 as is the length of the primary key in *EXAMPLE-V* above.

File Type W and R - Userview of Physical / Logical VSAM File

File types W (userview of physical VSAM file) and R (userview of logical VSAM files) are used to document DDMs for Natural for VSAM. DDMs documented with Predict objects of this type are used to access parts of the VSAM file record structure defined in the related physical/logical VSAM file.

The following rules apply:

- The relationship between a userview and physical VSAM fields is established by the two-character field attribute short name. Therefore field names can be changed in userviews and the connection to the corresponding field definition of a file remains.

- Only fields which are defined in the physical/logical VSAM file may be defined in the userview.

- The position of the field in the userview is independent from the VSAM file layout.

- Before a DDM can be generated from a file object of type W or R, the DDM of the corresponding physical/logical VSAM file must have been generated.

Generating DDMs from Predict VSAM Objects

When generating a DDM from a Predict object documenting a VSAM file, the file must already have been linked to a Predict database object of type V via "Contains FI". The database number is included in the DDM. The database number of this database must have been specified as a VSAM database in the Natural parameter module by the NTDB macro (e.g. NTDB VSAM,254).

Using Natural for VSAM with Physical VSAM Files

When a DDM generated from a VSAM file layout is used by Natural for VSAM, this DDM must always be available at runtime (it is not incorporated into the program at compile time).

Using a Record Layout Concept

Predict enables the use of different record layouts within a physical file (record layout concept) by the concept of logical VSAM files. When Natural for VSAM uses a DDM generated from a logical VSAM file only records with the VSAM prefix identifying that logical file will be returned.

When a logical VSAM file is used it is not necessary to define all fields in the physical VSAM file. Only the primary and alternate keys must be entered. The correct position of fields for keys must be ensured by insertion of DUMMY fields.

If Natural for VSAM is used in connection with logical VSAM files the rules outlined above have to be followed. This is especially true for the following point:

■ The VSAM prefix must be a fixed-length constant and it must precede the primary key in the logical file. Therefore the VSAM prefix plus primary key together in the logical file must have the same position and length as the primary key in the physical file.

Using a Record Layout Concept Without Logical VSAM Files

If the record type is not a constant or not the first part of the primary key, the logical VSAM files may not be used to generate DDMs for Natural for VSAM. In this case the following actions have to be taken if different record layouts are to be used in the same VSAM data set:

■ The layout of the different record type structures must be specified as multiple physical VSAM files containing the same DD name and therefore pointing to the same VSAM data set.

■ Check in the Natural program after the `FIND/READ` statement that the DDM corresponds to the record type. When a record does not correspond to the DDM, the record can be read again using the correct DDM. In these circumstances it is sometimes helpful to know the current record length. Natural for VSAM offers a subprogram which returns the record length.

6 Natural For DL/I

General Information

Natural for DL/I allows use of data stored in IMS/DL/I databases with Natural applications. Natural for DL/I uses the following control blocks:

- Natural for DL/I database descriptions (NDB) containing the information about the segment structure of an IMS/DL/I database and about the key fields of the segments.

- Natural for DL/I program specification blocks (NSB) reflecting an external view of a database, as it is used by an application program.

- User-defined fields (UDF) establishing a field structure in a database segment.

For more details see the description of the Natural SYSDDM Utility in the Natural Utilities Menu.

Predict supports the use of Natural for DL/I in the following ways:

- IMS/DL/I databases can be documented.

- User-defined fields can be documented (as segment layouts).

- Userviews of segments can be defined.

- Natural DDMs for IMS/DL/I segments and their userviews can be generated.

- Copy code for segment layouts in third generation languages can be generated.

- User-defined fields for Natural for DL/I can be generated.

The documentation of NSBs (Natural for DL/I program specification blocks) is currently not supported by Predict.

Documenting IMS/DL/I Data Structures

IMS/DL/I data structures are documented with objects of the following types:

- Databases are documented with database objects of type I.

- Segments are documented with file objects of type I.

- Sequence fields, search fields and alternate index fields are documented with field objects in these files.

- Segment layouts are documented with file objects of type J.

- Userviews are documented with file objects of type K.

Databases

There are two types of IMS/DL/I databases: physical and logical. The file list of a database object of type I consists only of files of type I.

Segments

Segments of an IMS/DL/I database are documented with file objects of type I. There are four types of IMS/DL/I segments:

- logical segments (only in logical databases);
- physical segments (only in physical databases);
- logical children (only in physical databases);
- virtual logical children (only in physical databases).

Each file of type I belonging to a physical database contains the sequence field, the search fields and the alternate index fields of the segment it documents. These fields are referred to below as "IMS/DL/I fields".

Each file of type I belonging to a logical database contains the IMS/DL/I fields of the segment of a physical database from which it is derived. A concatenated segment in a logical database contains the IMS/DL/I fields of both the logical child (virtual logical child) and the logical parent (physical parent of paired real logical child) from which it is derived.

Segment Layouts

User-defined layouts for an IMS/DL/I segment are documented with files of type J. Each file of type J has a master file of type I that documents the segment. Field definitions of a segment layout have the same structure as definitions of a sequential file: the position of a field cannot be specified directly but is determined by its offset. The offset is calculated from the lengths of the fields already defined. Therefore dummy fields must be defined if space is to be left free between two fields.

The IMS/DL/I fields of a file object of type I can be contained in the file of type J but they must have the same format, length and offset as in the file of type I.

Recommendations

Predict allows field IDs longer than 19 characters for files of type J; IDs of this length are not supported by SYSDDM. For this reason we recommend the following:

- Only use Predict to generate DDMs from files of this type. Do not use the utility SYSDDM. This can be enforced by setting the general default parameter Protection > SYSDDM utility to C or D.
- Only use the Predict Coordinator to transfer DL/I structures. If you use Natural utilities, field IDs longer than 19 characters will be truncated.

Userviews

Userviews of the segment are documented with file type K. Userviews have as master files the files of type I that document the segment. A userview (file type K) can contain the IMS/DL/I fields of the segment (type I) and fields of each layout (type J) of the segment.

Creating Objects for IMS/DL/I with Incorporation Functions

Databases and file objects of type I are created by incorporating Natural for DL/I NDBs using the Incorporate NDB function. These objects cannot be created manually using Predict maintenance functions Add Database/File.

The following rules apply for incorporation of IMS/DL/I databases and segments:

- A Natural for DL/I NDB is generated by assembling an IMS/DL/I database description (DBD) with the Natural for DL/I macro library according to the Natural Utilities documentation. When this has been done, the NDB can be incorporated into Predict.

- Before a logical NDB is incorporated, the physical NDB or NDBs from which the logical NDB is derived should be incorporated so that the references to source segments can be established. Also, if a physical NDB contains a virtual logical child and the paired real logical child is located in a different NDB, the NDB containing the real logical child should be incorporated first. If this is not possible, because there are either references back to the first NDB, or references to source segments inside the same NDB, the incorporation must be run twice to make sure that all source references are established.

- If user-defined fields for a segment have been defined in the SYSDDM DL/I services before the NDB is incorporated, the Incorporate NDB function incorporates the user-defined fields as well. In this case, at least one file of type J is created. If there are redefinitions in the user-defined fields, several layouts are created for the segment.

For details and options of the NDB incorporation function, see the section *Incorporation* in the *External Objects in Predict* documentation.

Maintaining Documentation for IMS/DL/I

The segment structure of an IMS/DL/I database and the format, length, offset and type of the IMS/DL/I fields can be changed by carrying out the following three steps:

- rewrite the IMS/DL/I database description
- reassemble the IMS/DL/I database description with the Natural for DL/I macro library
- incorporate the resulting NDB into Predict using the Replace option.

> **Note:** These attributes cannot be changed with maintenance functions as described in the section *Maintenance* in the *Predict Reference* documentation.

When an NDB is replaced, existing segment layouts in Predict are not replaced. Hence, user-defined fields are only incorporated once, and should from then on be maintained only in Predict.

Only certain attributes of Predict field objects contained in files of type I can be changed, for example, Field ID, Natural edit mask and Abstract. Certain changes to field formats are allowed, as described in the section *Field* in the *Predefined Object Types in Predict* documentation.

Maintaining Documentation of IMS/DL/I Segment Layouts

Segment layouts (type J) can be modified without restrictions. Also, new layouts can be created and existing layouts can be deleted.

Maintaining Documentation of IMS/DL/I Userviews

A userview (file of type K) can be created by selecting fields of a segment (file of type I) and fields of layouts (files of type J) that belong to that segment. Attributes such as field ID, Natural edit mask and field comments can be changed in Predict objects belonging to files of type K.

Generation Functions for Files of Type I, J and K

Generating DDMs

DDMs can be generated for files of type I, J and K.

The generation of a DDM requires the existence of the corresponding Natural for DL/I user-defined fields (UDF). The required UDF is generated automatically whenever a DDM for a segment is generated (or regenerated if the segment layouts have been changed). A UDF can also be generated independently from the generation of a DDM. When generating the UDF, Predict automatically selects a valid database number (for example a database number which is defined in an IMS or DL/I macro) and a free file number. These numbers are later used for the DDM generation.

The position and length of fields in a UDF is determined from the layouts of the segment (file type J).

Each DDM contains the IMS/DL/I fields of the given segment and the higher level segments. Additionally the following definitions will be contained for the different file types:

- The DDM of a file of type I contains the fields of all layouts of that segment.
- The DDM of a file of type J contains the fields of that layout.
- the DDM of a file of type K contains the fields of that userview.

Generating Copy Code

Copy code for record buffers in third generation programming languages can be generated for a given layout (file of type J). Synchronized and align options are not allowed: for FORTRAN copy code, fields must already lie within the appropriate boundary.

7 DB2 and SQL/DS

DB2 objects can be documented in Predict and generation, incorporation, comparison and administration functions can be applied to them.

> **Note:** To use functions of Predict that support DB2, Natural for DB2 or Natural SQL Gateway in a z/OS environment or Entire Access in a Windows, UNIX, Linux or OS/2 environment must be installed. Most functions described in this section apply both to DB2 and SQL/DS. Exceptions to this rule are listed as appropriate.

Documenting DB2 in Predict

The following topics are covered below:

- General Information
- Documenting DB2 Storagegroups or Database partition groups
- Documenting DB2 Databases
- Documenting DB2 Tablespaces and SQL/DS Dbspaces
- Documenting DB2 Tables and Views
- Documenting Referential Constraints
- Documenting DB2 Application Plans
- Documenting DB2 Packages
- Documenting DB2 Triggers
- Documenting DB2 Procedures and Functions
- Documenting Other DB2 Objects

General Information

DB2 storagegroups or database partition groups, databases, tablespaces, tables, views, columns, column masks, row permissions, distinct types, indexes, referential constraints, triggers, packages, application plans, procedures and functions can be documented in Predict.
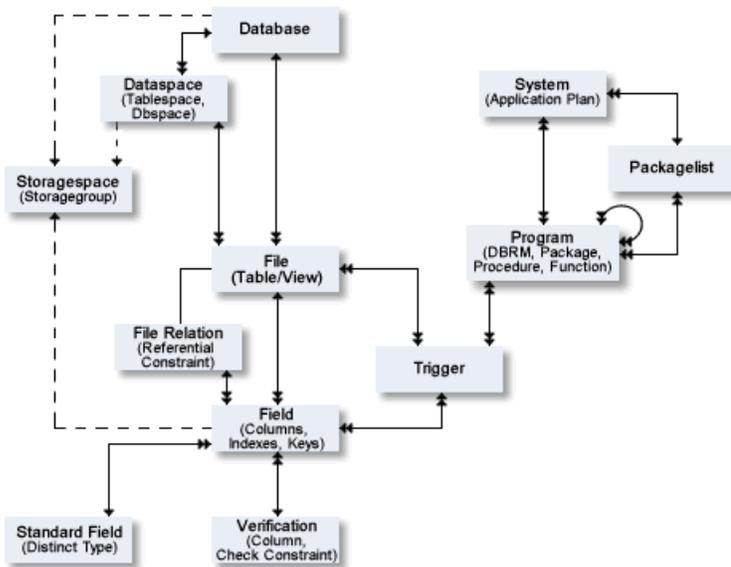
The following table gives an overview of how different DB2 objects are documented.

Documenting DB2 objects is described in the sections below.

DB2 Object	Documented in Predict with
Database	Database object of type D
Storagegroup (z/OS) / Database partition group (UNIX)	Storagespace object
DB2 tablespace / SQL/DS Dbspace	Dataspace object
Table / view	File object of type D or E
Table check constraint	Attribute of file object (type D)
Column check constraint	Verification object
Column mask	Access Definition object of type M
Row permission	Access Definition object of type M
Application plan (z/OS)	System object of type P
DBRM (z/OS)	Program object of type P and language Q
Package (z/OS)	Program object of type P and language B, C, F, H, P or Q or user-defined. Packages are linked to application plans with objects of type packagelist.
Collection (z/OS)	Packagelist attributes Collection name and Location name.
Index	Field attributes.
Column	Field objects.
Distinct Type	Field object of standard file *SAG-DISTINCT-TYPE*.
Trigger	Trigger object.

DB2 Object	Documented in Predict with
Procedure	Program object of type R.
Function	Program object of type U.

Documentation of multiple server

You are able to document several implementations of DB2 objects in different servers. A DB2 server is documented as a database object with Type S (SQL server).

With the means of SQL servers, it is possible to document one logical database design and to assign different physical database implementations to DB2 objects. The object types affected are Storagespace, Dataspace, Database, File, Field and Program. For objects of these types, the maintenance function Physical attributes is available to assign an individual setting of physical attributes to an object in a specific server.

For example, you may have two DB2 servers named DB2PROD and DB2TEST. It is then possible to define a tablespace named TABSPAC in the DB2PROD server as partitioned and in the DB2TEST server as simple or segmented tablespace.

There is one specific server named <Default Server>, that is always available and must not be documented. This server represents the DB2 server you are automatically connected to from your Natural or TP monitor environment on mainframe platforms. On a Windows or UNIX platform, the server defined for DBID 250 is used. Which real DB2 server is denoted by <Default Server> depends on your current environment.

When a DB2 object is added, it has a set of physical attributes in the <Default Server>. This set of attributes can be modified but can not be deleted. In the Add, Copy and Modify object function, you can edit attributes of the logical database design as well as attributes of the physical database design in one preferred server. You can specify your preferred server in the maintenance options of your profile. If your preferred server is not <Default Server>, the physical attributes for <Default Sever> will be taken from the general defaults when an object is added.

There are general defaults for DB2 objects such as Storagegroup, Database, Tablespace, Table, Index, Procedure and Function. You may adapt the defaults to meet the requirements of your enterprise. These default values will be preset and may be modified when an object is added.

When executing the retrieval model #S for an SQL server, all implementations of DB2 objects in this server are shown.

Generation, Administration, Incorporation and Comparison

If you are using IBM's Distributed Relational Database Architecture (DRDA) you will be able to connect to a specific DB2 server and perform the above mentioned functions for this specific server. This allows the administration of all DB2 servers in the network with one Predict installation. If you don't want to use explicitly documented servers, use <Default Server> to connect to your current DB2 server. When a function that has established a connection to a DB2 server other than

<Default Server> terminates, this connection is released and the connection to <Default Server> is reestablished. Therefore, the user will have the same database connection after leaving Predict, that he had when he started his Natural session.

As indicated above, it is possible to define an index as clustered in one server and as not clustered in a second server. This influences whether the component fields of the index may be updated or not. Whether a field may be updated or not is indicated by special short names in the DDM for Natural. In the DDM generation defaults you can specify the server that should be used for data access from Natural.

Support of DB2 in Natural for Windows and Natural for UNIX

Predict offers the support of DB2 also on Windows and UNIX platforms. To be able to access DB2, Natural for DB2 or Natural for SQL/DS is required on mainframe platforms. On Windows and UNIX platforms, Entire Access is required for access to DB2.

On Windows and UNIX platforms the DBID 250 is used to access DB2. DBID 250 must be defined as SQL in the Natural global configuration section. The connect string defined for DBID 250 is used to establish the connection to <Default Server>. After the execution of a function, that has established a connection to a DB2 server, the connection to <Default Server> is reestablished. It is therefore required, that the same user ID and password are defined for <Default Server> and every other DB2 server used.

When a connection to a server is established, a user ID and password may be required. The user ID and password can be changed within a Predict session by executing the command SET CATALOG_USER. This allows you to overwrite the previously entered user ID and password. This user ID and the password are then also used for every other connection to an SQL server.

Documenting DB2 Storagegroups or Database partition groups

Storagegroups or Database partition groups are documented as objects of type storagespace.

Documenting DB2 Databases

Databases are documented as objects of type database with database type D.

Only files of type D (DB2 table) can be linked to databases of type D.

> **Note:** DB2 databases on z/OS need only be documented with Predict database objects if you intend to generate the DB2 database from the Predict database object. If you use the option to create the DB2 database implicitly when generating tablespaces, tables or views, the database need not be documented with a Predict database object.

Documenting DB2 Tablespaces and SQL/DS Dbspaces

In DB2/SQL/DS, tables/views are not directly linked to databases: a DB2 tablespace or SQL/DS Dbspace establishes the connection of tables/views and databases.

DB2 tablespaces and SQL/DS Dbspaces are documented with Predict dataspace objects of type D (DB2) or S (SQL/DS) in a z/OS environment and type 2 in a UNIX environment.

Notes:

1. DB2 tablespaces on z/OS need only be documented with Predict dataspace objects if you intend to generate the DB2 tablespace from the Predict dataspace object. If you use the option to create the DB2 tablespace implicitly when generating tables/views, the tablespace need not be documented with a Predict dataspace object. Partitioned or segmented tablespaces are not created implicitly.

2. DB2 tablespaces in a UNIX environment need only be documented with Predict dataspace objects if you intend to generate the DB2 tablespace from the Predict dataspace object. If you use the option to use IBMDEFAULTGROUP when generating tables/views, the tablespace need not be documented with a Predict dataspace object.

An SQL/DS Dbspace must be documented with a Predict tablespace object because a Dbspace cannot be created implicitly.

No auxiliary tablespaces are supported. See *Columns With Format LOB* for further information.

For tablespaces with partition type G (Partition by growth) it is possible to specify the number of partitions to be initialized with the function Partitions initialized. This function can be set in the Generation Defaults for DB2 tablespaces.

Refer to the parameter description given in DB2 Tablespace of the section *Generation of External Objects* in the *External Objects in Predict* documentation for details.

Documenting DB2 Tables and Views

Tables are documented as files of type D. Views are documented as files of type E.

Note: If a table contains a partitioning index, the number of partitions must be documented as an attribute of the file if the file is not linked to a dataspace via association "Contains FI".

Subselect Clauses and Expressions in Field Definitions

The documentation of views is supported by an additional editor to specify the part of the subselect clause starting from the first `FROM` clause.

The selection clause of the subselect clause is documented by the specification of the field list of the view. The specified list of tables/views in the first `FROM` clause of the subselect clause is generated

by Predict and will be updated if a field from an additional table/view is added to the view. Correlation names can be added to the tables and views in the list (using editor functions). The remaining part of the subselect clause is left unchanged.

The expression used to define DB2 or SQL/DS fields can contain complex expressions. Fields that are defined not only by a single column name but use either a constant or a more complex expression are called derived fields. A special editor is provided for specifying the expression of derived fields.

In the field expression and in the subselect clause, comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

DB2 offers the possibility to use subqueries and joined tables in the `FROM` clause of the view definitions. This functionality is also supported under Predict.

You can define in the subquery of a view whether the `FROM` clause is made up of an `EXCEPT`, `EXCEPT ALL`, `INNER`, `INTERSECT`, `INTERSECT ALL`, `LEFT OUTER`, `RIGHT OUTER`, `FULL OUTER JOIN`, `UNION` or `UNION ALL`. Use the subquery editor to add and modify joined views.

Predict has a file of type IV (Intermediate View) that enables you to use subselects in the `FROM` clause of a view definition. Files of type IV have a field list to show the selection clause and a subquery to show the search condition. Just like views, files of type IV can have a Join type. They can be used as master files for views and files of type IV. When a view is generated that has a file of type IV as a master file, a `SELECT` partial statement is generated into the `FROM` clause. Files of type IV are not in the DB2 catalog. Nevertheless, the same naming standards are valid for files of type IV as for tables and views. See *Naming Conventions for DB2* for further information.

When using Incorporate and Compare, files of type IV are created in addition to the view in order to represent subselects in the `FROM` clause. Files of type IV are implemented as file objects in Predict, in order to ensure the consistency of the field definitions via rippling.

The maximum number of master files for a view or file of type IV is 100 in Predict.

Common Table Expression

A common-table-expression used in the definition of a DB2 view or a materialized query table is documented by means of an intermediate view (file with file type IV). This intermediate view can then be referenced in the WITH clause of the subquery.

Assume the following example view definition in DB2:

```
CREATE VIEW SMR.FIRST_CTE_VIEW AS
WITH DTOTAL (DEPTNO, TOTALPAY) AS
(SELECT WORKDEPT, SUM(SALARY+BONUS)
FROM  DSN8810.EMP
GROUP BY WORKDEPT)
SELECT DEPTNO
FROM DTOTAL
WHERE TOTALPAY = (SELECT MAX(TOTALPAY) FROM  DTOTAL)
```

The common table expression is documented as intermediate view named SMR-FIRST_CTE_IV_1.

The subquery editor of the DB2 view SMR.FIRST_CTE_VIEW looks like this:

```
WITH
    SMR-FIRST_CTE_IV_1
FROM
    SMR-FIRST_CTE_IV_1
WHERE SMR-FIRST_CTE_IV_1-TOTALPAY = (SELECT MAX(TOTALPAY) FROM
SMR-FIRST_CTE_IV_1)
```

And the subquery editor of SMR-FIRST_CTE_IV_1 looks like this:

```
FROM
    DSN8810-EMP
GROUP BY DSN8810-EMP-WORKDEPT
```

The field TOTALPAY of file SMR-FIRST_CTE_IV_1 has field type DV (derived field) and has the expression:

```
SUM(DSN8810-EMP-SALARY+DSN8810-EMP-BONUS)
```

> **Note:** In DB2 common table expressions can be used to create recursive SQL. Views based on such recursive definitions can not be incorporated or compared. It is also not possible to document and generate such views.

Temporal Tables

DB2 supports so-called temporal tables. A temporal table is a table that records the period of time when a row is valid.

DB2 supports two types of periods, which are the system period (SYSTEM_TIME) and the application period (BUSINESS_TIME).

A system-period temporal table is a base table that is defined with system-period data versioning. You can modify an existing table to become a system-period temporal table by specifying the *ADD PERIOD SYSTEM_TIME* clause on the `ALTER TABLE` statement. After creating a history table that corresponds to the system-period temporal table, you can define system-period data versioning on the table by issuing the `ALTER TABLE ADD VERSIONING` statement with the *USE HISTORY* table clause.

An application-period temporal table is a base table that includes an application period (BUSINESS_TIME). You can modify an existing table to become an application-period temporal table by specifying the *ADD PERIOD BUSINESS_TIME* clause on the `ALTER TABLE` statement.

A bi-temporal table is a table that is both a system-period temporal table and an application-period temporal table. You can use a bi-temporal table to keep application period information and system-based historical information.

In Predict, a temporal table is documented as a file of type D using specific superfields. These fields are named either SYSTEM_TIME or BUSINESS_TIME, depending on the type of temporal table.

For system-period temporal tables it is required to specify a field of type TS to be used `AS TRANSACTION START ID`. This field specifies that a timestamp value is assigned when the row is inserted or any column in the row is updated. A table can have only one transaction-start-id column.

Archive-enabled and Archive Tables

DB2 supports so-called archive tables. An archive table is a table that stores older rows from another table. The original table is called an archive-enabled table.

DB2 can automatically store rows that are deleted from an archive-enabled table in an associated archive table. When you query an archive-enabled table, you can specify whether you want those queries to include data from the archive table.

In Predict, an archive table is documented as a file of type D using specific attributes to link the archive table to a specific base table and also to define how the archive table is maintained.

Hash key

When you create or alter a table, you can organize the table by hash to improve the performance queries that access individual rows.

When you create or alter tables on universal table spaces, you can enable hash access to that table by adding the `organization-clause` to your `CREATE TABLE` or `ALTER TABLE` statement.

In Predict you can document hash-organized tables with files of type D or MT. You have to specify a hash size that will be preallocated for an associated partition.

If hash size is specified it is also required to specify a field as hash key.

Refer to your *DB2* documentation for further information on hash-organized tables.

Clone Tables

DB2 supports so-called clone tables. A clone table is a table that has the exact same attributes as an already existing base table.

- In Predict, a clone table is documented as a file of type D. Base and clone table are documented in Predict as separate files of type D which allows the creation of an individual DDM for each file.

- Clone tables can be incorporated, compared and dropped using the Incorporate DB2 Table/View and Compare DB2 Table/View functions and purged using option P of the External File Administration function.

- The Predict maintenance does not guarantee that the individual files of type D that represent the base and the clone table in Predict have identical field lists. The user has to make the required field list checks.

- The Generate DB2 Table/View function does not allow the modification of field lists of existing cloned DB2 tables. This restriction also applies to the corresponding base table documented in Predict.

- Predict does not create clone tables nor does Predict generate an `EXCHANGE DATA` statement. Such statements have to be submitted by the user from outside Predict (for example, using SYSDB2 or any other tool that can issue interactive SQL statements).

Documenting Referential Constraints

Referential constraints are documented as file relations of type R (referential constraint). The relation is established between a unique constraint and a foreign key. Unique constraint and foreign key can belong to the same or to different tables.

Documenting DB2 Application Plans

DB2 application plans are documented as Predict system objects of type P.

Documenting DB2 Packages

With DB2 Version 2.3 or above, DBRMs can be grouped to packages. Packages are linked to plans dynamically (at run-time). DB2 packages are documented in Predict with program objects of with language B, C, F, H, P or Q or user-defined.

Linking Packages to Application Plans with Packagelists

Packages are linked to application plans with objects of type packagelist (PG). The subtypes of the object type packagelist determine the type of inclusion of packages into an application plan when binding a plan. Valid values are:

- **Q**
  DBRM

- **T**
  total collection

- **S**
  subcollection

The different subtypes are described in the section *Using Predict Information when Binding Application Plans*.

Packages are referenced in a plan by collections. Any collection is a virtual summary of packages, used to simplify references to packages. Any package can be contained in several collections. Collections are documented as attributes of packagelists.

Documenting DB2 Triggers

Trigger objects represent DB2 triggers.

You can link an unlimited number of triggers to a table. Update triggers that can only be executed if certain table fields are changed are then linked to these fields. The appropriate update-clauses are created during the table generation process. All triggers must be linked to file.

Using Incorporate and Compare on the tables creates the trigger objects in Predict and establishes the links to the file and to the fields. The connection to the DB2 catalog is not the Predict object ID, but an attribute Triggername. See *Naming Conventions for DB2* for further information.

In addition, triggers contain information as to when they are executed (during Insert, Update or Delete) and whether they are to be executed before or after a certain statement is executed. The code that is to be executed is noted in the trigger body, which is a text attribute in Predict. The use of procedures in triggers is retrieved from the text of the trigger body.

If two or more triggers contained in a trigger list are assigned the same trigger action (for example Insert), trigger type (for example Before) and event (the code that is to be executed) then the trigger positioned first in the trigger list will be executed first. The following triggers will then be executed according to their position in the trigger list.

Documenting DB2 Procedures and Functions

DB2 gives you the possibility to write procedures and user-defined functions. These objects can be documented as programs in Predict. The program type R (SQL procedure) can accept many specifications that are only related to DB2. Procedures can be implemented in third generation programming languages or SQL.

DB2 functions are documented as Programs of type U (Database function) in Predict. These functions can return a table as a result. There are two associations between PR and FI with the names "Input FI" and "Returns FI". The association names are supposed to indicate that the linked files represent the structure of the input parameter or the structure of the results table.

Only files of type IT (Intermediate Table) can be linked (see below). If the entered value or the results table is a scalar value, then you still must create a file that has exactly one field. In DB2, you can use table functions in the definition from views. Predict does not support this.

Files of Type IT

There is a new file type IT (Intermediate Table) for documenting the formal parameters of database functions. They do not exist in DB2. Their fields can, like tables, have a link to the standard file *SAG-DISTINCT-TYPE*, that is interpreted as being distinct type.

Documenting Other DB2 Objects

DB2 Distinct Types

In Predict, distinct types are implemented with the help of an indicated standard file called *SAG-DISTINCT-TYPE*.

The connection with the DB2 catalog is established by the names of the fields of this standard file. The field names of *SAG-DISTINCT-TYPE* consist of a SCHEMA_NAME and a TYPE_NAME. See *Naming Conventions for DB2* for further information. Table fields that are connected to a standard field in *SAG-DISTINCT-TYPE* have the predefined format which is the basis for the type. Changes in the type definition are spread via rippling to the derived fields.

> **Note:** The name of the standard field is not a valid column name. After copying from *SAG-DISTINCT-TYPE* via the `SEL` command into the field list of a table, the field name must be changed so that it conforms with the SQL naming standards.

When a table is generated, a `CREATE DISTINCT TYPE` statement is created for every type in the fields with distinct type, if the type has not already been defined in the DB2 catalog. The type definition of the table field is in this case the distinct type name.

When using the commands Incorporate and Compare on the table, the connections to the type definition are also compared. If the type definition in Predict is different from the one in DB2, then the field format of the table field is adapted to the catalog entry, and the field is marked as NON-Standard.

The type definitions are always regarded by the tables using it. Therefore, there are no explicit Generate, Incorporate and Compare distinct type functions. A type definition in DB2 is also deleted via Drop if the last table using it is dropped by Administration (Database, dataspace or file). Since every distinct type is based on a predefined type, the table fields derived from these types are represented in the DDM with the predefined type.

Arrays

Arrays can be used as parameters or return values of functions. When generating such a function, the appropriate type definitions will also be generated.

Array types are documented as multiple value fields in file *SAG-DISTINCT-TYPE*. The occurrences can be used to define the dimension of the array. To use associative arrays that are indexed by values of a second data type, link a field of file *SAG-DISTINCT-TYPE* that serves as index type definition into association "Indexed by" to the array type field.

DB2 Columns

DB2 columns are documented as field objects.

Columns With Format LOB

LOBs are represented as the field format LO. The character set determines whether it is a BLOB, CLOB or a DBCLOB. The lengths of these fields can be declared in the following units: bytes, kilobytes, megabytes or gigabytes.

In order to create a connection between a row in the original table and a LOB value, DB2 uses so-called auxiliary tablespaces, auxiliary tables and an index to save the LOB values. Predict does not support the creation of these database objects.

Instead Predict uses the DB2 feature that automatically create these auxiliary objects. This is achieved by generating a statement `SET CURRENT RULE='STD'` whenever a table with LOB column is to be generated, provided that the Special Register Current Rule does not already have this setting.

It is possible to provide an inline length integer value for BLOB and CLOB columns. It specifies the maximum number of bytes that are stored in the base table space for the column.

The inline length can be set in the *Base Extensions* screen available for fields of type D with format LO.

Columns With Format ROWID

ROWID fields are documented with fieldtype QN. Their format is A and their length is 40. The field maintenance ensures that only one ROWID field exists per table. It also ensures that every table containing a LOB column also contains a ROWID column. This is necessary, so that DB2 can create the index to connect the row in the original table with the auxiliary table.

When deleting databases, tablespaces and tables, the auxiliary objects are deleted as well. It is possible to define an identity property for numeric fields. The contents of these fields can be generated by DB2. This is an easy way to created a primary key.

LOB fields are skipped during the DDM generation process, since NAT 3.x does not make any dynamic variables available. ROWID fields are only represented by A40 fields in the DDM.

Column Masks

A column mask in DB2 is used for column access control and specifies the value that should be returned for a specified column. Exactly one mask per column is allowed.

In DB2 an enabled column mask does not take effect until the ALTER TABLE statement with the ACTIVATE COLUMN ACCESS CONTROL clause is used to activate column access control for the table.

Column masks for DB2 fields are documented with Access definitions of type M.

If a DB2 table documented in Predict uses column masks and the table is dropped, the corresponding column masks are also deleted in Predict.

Row Permissions

Multiple row permissions can be created for a table.

In DB2 an enabled row permission does not take effect until the ALTER TABLE statement with the ACTIVATE ROW ACCESS CONTROL clause is used to activate row access control for the table.

Row permissions for files of type D are documented with Access definitions of type P.

If a DB2 table documented in Predict uses row permissions and the table is dropped, the corresponding row permissions are also deleted in Predict.

DB2 Indexes

DB2 indexes are documented with field objects as follows:

- Field attributes
  - index name
  - definition of index
  - using- and free-block
- If the index consists of only *one* column, the field documenting the column is marked as a descriptor with descriptor type D, P or F.
- If the index consists of *multiple* columns, it is documented as a field with field type SP (superfield) and descriptor type D, P or F. The descriptor types have the following meaning:
  - **D**
    Field is an index.
  - **F**
    Field is a foreign key and an index.
  - **P**
    Field is a primary key. This always implies that the field is also a unique index.

Unique Constraints

Unique constraints are documented as follows:

- If the unique constraint applies to only *one* column, the field documenting the column is marked U in column Unique option.
- If the unique constraint applies to *multiple* columns, it is documented as a field of type SP (superfield) with descriptor type D, F or P, and U in column Unique option. The descriptor types have the following meaning:

- **D**

  Field is a unique index.

- **F**

  Field is a foreign key with unique index.

- **P**

  Field is a primary key. This always implies that the field has a unique constraint.

Foreign Keys

Foreign keys are documented as follows:

- If the foreign key consists of only one column, the field documenting the foreign key is marked as a descriptor with descriptor type F or E.

- If the foreign key consists of multiple columns, it is documented as a field with field type SP (superfield) and descriptor type F or E. The descriptor type F means the field is a foreign key and an index. E means the field is a foreign key without an index.

Column Check Expressions

Check expressions for single columns are documented with verifications of status SQL. The check expression is stored as the rule of the verification.

Check expressions can be edited with the Predict Rule Editor.

Comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

Table Check Expression

A table check expression is a check expression that applies to more than one column. A table check expression is an attribute of a file.

To edit table check expressions, enter Y in the field Check expression of the corresponding file object.

Comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

Naming Conventions for DB2

DB2 naming conventions must be observed when creating or maintaining Predict objects for DB2. The following rules apply:

- General Rules
- Storagegroup Name
- Collection Name
- Constraint Name

- Correlation Name
- Distinct Type
- Procedure Name
- Index Name
- Table/View Name
- Function Name
- Column Name
- Trigger Name
- Delimited Identifier

General Rules

Valid identifiers are from 2 to 32 characters long, must start with an alpha character (A - Z) and may be followed by either an alpha, US national character (#, $, @), a digit or underscore character. Identifiers must comply with these rules if Predict maintenance and generation functions are to be applied.

Concatenated names must not consist of more than 32 characters including the hyphen. See the following paragraphs for details.

Hyphens in names are treated as follows:

- When a table/view is generated from a Predict file object, the hyphen will be transformed into a period (.).
- Because hyphens are used as delimiters, only one hyphen can occur in the file ID.
- The hyphen can be used as a minus sign or negative sign in the field expression or the subselect clause and must then be preceded by a blank.
- Column names must not contain a hyphen.

> **Notes:**

1. Incorporation functions accept object and creator names as selection criteria according to the restrictions mentioned here and in the following sections. DB2 objects with a name longer than the described ones are not offered for incorporation.

2. Names that apply to the restrictions mentioned here and in the following sections but can not be shown in full length are indicated by ">" as last character.

3. For views and intermediate tables use the editor commands `FLIP C` and `FLIP T` to edit the field names and their derivation in full length.

Storagegroup Name

Storagegroup names can consist of up to 18 characters.

Collection Name

Collection names can consist of up to 18 characters.

Constraint Name

Contraint names can consist of up to 32 characters.

Correlation Name

Correlation names can be defined in the subselect clause of a view. If a correlation name is defined for a table/view in the subselect clause, all references (in field expressions as well as in the field editor of the view) to columns of the table/view must be qualified with the correlation name. If no correlation name is defined for a table/view in the subselect clause, all references to columns of the table/view must be fully qualified with creator-tablename-columnname (for example: SYSIBM-SYSCOLUMNS-COLNAME).

Correlation names can consist of up to 18 characters.

Distinct Type

Distinct types consist of SCHEMA_NAME and TYPE_NAME concatenated by a hyphen (used as qualification character).

SCHEMA_NAME and TYPE_NAME each can consist of up to 32 characters, but a concatenation of both (including the hyphen as qualification character) must not exceed a length of 32 characters.

Procedure Name

Procedure names must be defined in unqualified form (a long SQL identifier). Procedure names can consist of up to 32 characters.

Index Name

Index names consist of the creator name and the actual name of the index concatenated by a hyphen (used as qualification character).

Creator name and actual name of the index each can consist of up to 32 characters, but a concatenation of both (including the hyphen as qualification character) must not exceed a length of 32 characters.

Table/View Name

Table/View names consist of the creator name and the actual name of the table/view concatenated by a hyphen (used as qualification character).

Creator name and actual name of the table/view each can consist of up to 32 characters, but a concatenation of both (including the hyphen as qualification character) must not exceed a length of 32 characters.

Function Name

Function names must be defined in unqualified form (a long SQL identifier). Function names can consist of up to 32 characters.

Column Name

Column names can consist of up to 30 characters.

Trigger Name

Trigger names consist of the creator name and the actual name of the trigger concatenated by a hyphen (used as qualification character).

Creator name and actual name of the trigger each can consist of up to 32 characters, but a concatenation of both (including the hyphen as qualification character) must not exceed a length of 32 characters.

Delimited Identifier

With DB2 or SQL/DS, special characters can be used in identifiers of tables and views. Identifiers that contain special characters have to be delimited (usually with single or double quotes) and are therefore called delimited identifiers.

DB2 or SQL/DS tables and views with delimited identifiers can be incorporated. They can then be renamed with Predict maintenance functions and retrieval functions can be applied to them. It is strongly advisable to rename delimited identifiers for the following reasons:

- The only Predict functions that can be applied without restriction to objects with delimited identifiers are Incorporate and Rename.

- If identifiers contain special characters such as blank or asterisk, results of retrieval functions are unpredictable.

- Views can only be generated if the subselect clause and the column expressions do not contain references to delimited identifiers enclosed by quotation marks.

As the SQL escape character Predict uses quotation marks (") and as the SQL string delimiter apostrophes (') are used. The Predict Incorporate function converts other escape characters or string delimiters to quotation marks (") and apostrophes (').

Generating, Incorporating and Comparing DB2 Objects

Prerequisites

Generation, Incorporation and Comparison are subject to DB2 security mechanisms. The following mechanisms apply for DB2 on z/OS:

- To perform the Generate function and administration functions Purge and Refresh, the user must have the appropriate privileges within DB2 / SQL/DS.

- To perform Incorporation and Comparison functions, the user must have SELECT privilege on nearly all catalog tables.

  - The SELECT privilege in DB2 is the minimum prerequisite for incorporation of DB2 tables/Views (see *GRANT (TABLE or VIEW PRIVILEGES)* in your DB2 documentation for the following tables:

    SYSIBM-SYSCHECKDEP
    SYSIBM-SYSCHECKS
    SYSIBM-SYSCOLUMNS
    SYSIBM-SYSDATABASE
    SYSIBM-SYSDATATYPES
    SYSIBM-SYSDUMMY1

SYSIBM-SYSFIELDS
SYSIBM-SYSFOREIGNKEYS
SYSIBM-SYSINDEXES
SYSIBM-SYSINDEXPART
SYSIBM-SYSKEYCOLUSE
SYSIBM-SYSKEYS
SYSIBM-SYSPLAN
SYSIBM-SYSPARMS
SYSIBM-SYSSTOGROUP
SYSIBM-SYSRELS
SYSIBM-SYSROUTINES
SYSIBM-SYSSEQUENCES
SYSIBM-SYSSEQUENCESDEP
SYSIBM-SYSTABCONST
SYSIBM-SYSTRIGGERS
SYSIBM-SYSSYNONYMS
SYSIBM-SYSTABLEPART
SYSIBM-SYSTABLES
SYSIBM-SYSTABLESPACE
SYSIBM-SYSVIEWDEP
SYSIBM-SYSVIEWS
SYSIBM-SYSVOLUMES
SYSIBM-SYSDATATYPES
SYSIBM-SYSTRIGGERS

- To use the SQL statements generated by Predict, the corresponding DB2 privileges are also required.

- To incorporate the SQL/DS tables contained in the following list, the SELECT privilege in SQL/DS is also a prerequisite:

SYSTEM-SYSCATALOG
SYSTEM-SYSCOLUMNS
SYSTEM-SYSDBSPACES
SYSTEM-SYSFIELDS
SYSTEM-SYSINDEXES
SYSTEM-SYSKEYCOLS
SYSTEM-SYSKEYS
SYSTEM-SYSSYNONYMS
SYSTEM-SYSUSAGE
SYSTEM-SYSVIEWS

The following mechanisms apply for DB2 on UNIX:

- To perform the Generate function and administration functions Purge and Refresh, the user must have the appropriate privileges within DB2 / SQL/DS.

- To perform Incorporation and Comparison functions, the user must have SELECT privilege on nearly all catalog tables.

  - The SELECT privilege in DB2 is the minimum prerequisite for incorporation of DB2 tables/Views (see *GRANT (TABLE or VIEW PRIVILEGES)* in your DB2 documentation for the following tables:

    SYSCAT-BUFFERPOOLS
    SYSCAT-CHECKS
    SYSCAT-COLCHECKS
    SYSCAT-COLUMNS
    SYSCAT-CONSTDEP
    SYSCAT-DATATYPES
    SYSCAT-INDEXCOLUSE
    SYSCAT-INDEXES
    SYSCAT-KEYCOLUSE
    SYSCAT-NODEGROUPDEF
    SYSCAT-NODEGROUPS
    SYSCAT-REFERENCES
    SYSCAT-ROUTINEDEP
    SYSCAT-ROUTINEPARMS
    SYSCAT-ROUTINES
    SYSCAT-SEQUENCES
    SYSCAT-TABCONST
    SYSCAT-TABLES
    SYSCAT-TABLESPACES
    SYSCAT-TRIGGERS
    SYSCAT-VIEWDEP
    SYSCAT-VIEWS

- To use the SQL statements generated by Predict, the corresponding DB2 privileges are also required.

Generation

DB2 objects can be generated from Predict documentation objects.

DB2 database	The function is not available for SQL/DS.
	The function is not available for DB2 on UNIX.
	Command: `GENERATE DB2-DATABASE`
DB2 storagegroup	Command: `GENERATE STORAGEGROUP`
DB2 database partition group	
DB2 tablespace	Command: `GENERATE TABLESPACE`
SQL/DS Dbspace	

DB2/SQL/DS table/view	Columns, indexes, referential constraints, triggers and distinct types are automatically included when generating DB2 tables and views.
	Command: `GENERATE TABLE`

Rules Applying when Generating DB2 / SQL/DS Objects

- All objects are generated by first generating the SQL statements that are necessary to implement the object and then executing these statements. An additional confirmation is requested before a DB2 object is actually implemented.

- The generated SQL statements can be saved in a protocol.

- If a generation function is executed for an object that is already implemented, the existing DB2 object can be updated.

- Tables with LOB column: Statement `SET CURRENT RULE='STD'` is created when a table with LOB column is generated.

- Auxiliary tablespaces, tables and indexes are created automatically by DB2.

See respective sections in *Generation* in the *External Objects in Predict* documentation for more information.

Incorporation

The incorporation functions create Predict documentation objects for databases (not for SQL/DS), tablespaces, storagegroups, tables/views, (including columns, indexes and referential constraints) from the system catalog.

See the section *Incorporation* in the *External Objects in Predict* documentation.

Comparison

The comparison functions list differences between the current implementation in DB2 / SQL/DS and the corresponding documentation. The documentation can be updated to match the implementation.

See the section *Comparison* in the *External Objects in Predict* documentation.

Administrating Implemented DB2 Objects

Functions for administrating DB2 objects are provided to display, purge or refresh DB2 objects that have been implemented from Predict documentation.

- The Display function lists generation protocols.

- The Purge function drops a table/view physically in DB2 or SQL/DS. If a table holds the last reference to a distinct type, the distinct type is also dropped.

- The Refresh function deletes all data in an implemented table but keeps the table structure.

See the section *Administration of External Objects* in the *External Objects in Predict* documentation.

Locking the Functions of the DB2 Utilities SYSDB2 and SYSSQL

With the Natural for DB2 utilities SYSDB2 (for DB2) and SYSSQL (for SQL/DS) storagegroups, databases, tablespaces/Dbspaces, tables/views and indexes can be created or modified. To avoid undocumented changes to DB2 or SQL/DS concerning these object types, your data dictionary administrator (DDA) may have set the parameter SYSDB2 utility in the Defaults > General Defaults > Protection screen.

A	Allowed: all SYSDB2 functions can be executed.

D	Disallowed: the following SYSDB2 functions cannot be executed:
	
	`CREATE DATABASE`
	`CREATE STORAGEGROUP`
	`CREATE TABLE`
	`CREATE TABLESPACE`
	`CREATE VIEW`
	`CREATE INDEX`

I	Incorporate: all SYSDB2 functions can be executed outside of Predict. If one of the following statements is submitted to DB2, an automatic incorporation in Predict is performed:
	
	`CREATE DATABASE`
	`CREATE STORAGEGROUP`
	`CREATE TABLE`
	`CREATE TABLESPACE`
	`CREATE VIEW`

8 Oracle

Oracle objects can be documented in Predict and generation, incorporation, comparison and administration functions can be applied to them.

> **Note:** To use functions of Predict that support Oracle in a Linux, UNIX or Windows environment, Entire Access in a Linux, UNIX or Windows environment must be installed.

Documenting Oracle in Predict

The following topics are covered below:

- General Information
- Documenting Oracle Tables and Views
- Documenting Referential Constraints
- Documenting Other Oracle Objects

General Information

Oracle tables, views, columns, indexes and referential constraints can be documented in Predict.
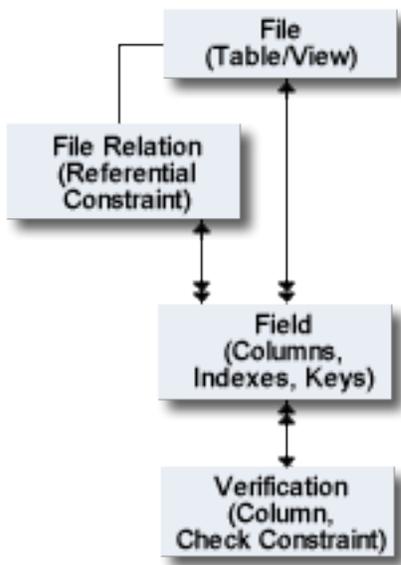
The following table gives an overview of how different Oracle objects are documented.

Documenting Oracle objects is described in the sections below.

Oracle Object	Documented in Predict with
Table / view	File object of type OT or OV
Table check constraint	Attribute of file object (type OT)
Column check constraint	Verification object
Index	Field attributes.
Column	Field objects.

Documentation of multiple server

You are able to document several implementations of Oracle objects in different servers. An Oracle server is documented as a database object with Type O (Oracle handler).

With the means of Oracle handlers, it is possible to document one logical database design and to assign different physical database implementations to Oracle objects. The object types affected are File and Field. For objects of these types, the maintenance function Physical attributes is available to assign an individual setting of physical attributes to an object in a specific server.

There is one specific server named <Unassigned>, that is always available and must not be documented. This server is used for all objects not assigned to a documented Oracle handler.

When a Oracle object is added, it has a set of physical attributes in the <Unassigned>. This set of attributes can be modified but can not be deleted. In the Add, Copy and Modify object function, you can edit attributes of the logical database design as well as attributes of the physical database design in one preferred server. You can specify your preferred server in the maintenance options of your profile. If your preferred server is not <Unassigned>, the physical attributes for <Unassigned> will be taken from the general defaults when an object is added.

There are general defaults for Oracle objects such as Table and Index. You may adapt the defaults to meet the requirements of your enterprise. These default values will be preset and may be modified when an object is added.

Generation, Administration, Incorporation and Comparison

As indicated above, it is possible to define an index as partitioned in one server and as not partitioned in a second server.

Support of Oracle in Natural for Windows and Natural for UNIX

Predict offers the support of Oracle also on Windows and UNIX platforms. On Windows and UNIX platforms, Entire Access is required for access to Oracle.

When a connection to a server is established, a user ID and password may be required.

Documenting Oracle Tables and Views

Tables are documented as files of type OT. Views are documented as files of type OV.

> **Note:** If a table is partitioned, the number of partitions must be documented as an attribute of the file.

Subselect Clauses and Expressions in Field Definitions

The documentation of views is supported by an additional editor to specify the part of the subselect clause starting from the first `FROM` clause.

The selection clause of the subselect clause is documented by the specification of the field list of the view. The specified list of tables/views in the first `FROM` clause of the subselect clause is generated by Predict and will be updated if a field from an additional table/view is added to the view. Correlation names can be added to the tables and views in the list (using editor functions). The remaining part of the subselect clause is left unchanged.

The expression used to define Oracle fields can contain complex expressions. Fields that are defined not only by a single column name but use either a constant or a more complex expression are called derived fields. A special editor is provided for specifying the expression of derived fields.

In the field expression and in the subselect clause, comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

Oracle offers the possibility to use subqueries and joined tables in the `FROM` clause of the view definitions. This functionality is also supported under Predict.

You can define in the subquery of a view whether the `FROM` clause is made up of an `EXCEPT`, `EXCEPT ALL`, `INNER`, `INTERSECT`, `INTERSECT ALL`, `LEFT OUTER`, `RIGHT OUTER`, `FULL OUTER JOIN`, `UNION` or `UNION ALL`. Use the subquery editor to add and modify joined views.

Predict has a file of type IV (Intermediate View) that enables you to use subselects in the `FROM` clause of a view definition. Files of type IV have a field list to show the selection clause and a subquery to show the search condition. Just like views, files of type IV can have a Join type. They can be used as master files for views and files of type IV. When a view is generated that has a file of type IV as a master file, a `SELECT` partial statement is generated into the `FROM` clause. Files of type IV are not in the Oracle catalog. Nevertheless, the same naming standards are valid for files of type IV as for tables and views. See *Naming Conventions for Oracle* for further information.

When using Incorporate and Compare, files of type IV are created in addition to the view in order to represent subselects in the `FROM` clause. Files of type IV are implemented as file objects in Predict, in order to ensure the consistency of the field definitions via rippling.

The maximum number of master files for a view or file of type IV is 100 in Predict.

Archive Tables

Oracle supports so-called archive tables. An archive table is a table that stores older rows from another table.

Oracle can automatically store rows that are deleted from a table in an associated archive table.

Documenting Referential Constraints

Referential constraints are documented as file relations of type R (referential constraint). The relation is established between a unique constraint and a foreign key. Unique constraint and foreign key can belong to the same or to different tables.

Documenting Other Oracle Objects

Oracle Columns

Oracle columns are documented as field objects.

Columns With Format LOB

LOBs are represented as the field format LO. The character set determines whether it is a BLOB, CLOB or a DBCLOB. The lengths of these fields can be declared in the following units: bytes, kilobytes, megabytes or gigabytes.

Columns With Format ROWID

ROWID fields are documented with fieldtype QN. Their format is A. The field maintenance ensures that only one ROWID field per table exists.

It is possible to define an identity property for numeric fields. The contents of these fields can be generated by Oracle. This is an easy way to create a primary key.

Oracle Indexes

Oracle indexes are documented with field objects as follows:

- Field attributes
    - index name
    - definition of index
    - index attributes
- If the index consists of only *one* column, the field documenting the column is marked as a descriptor with descriptor type D, P or F.
- If the index consists of *multiple* columns, it is documented as a field with field type SP (superfield) and descriptor type D, P or F. The descriptor types have the following meaning:

- **D**

  Field is an index.

- **F**

  Field is a foreign key and an index.

- **P**

  Field is a primary key. This always implies that the field is also a unique index.

Unique Constraints

Unique constraints are documented as follows:

- If the unique constraint applies to only *one* column, the field documenting the column is marked U in column Unique option.

- If the unique constraint applies to *multiple* columns, it is documented as a field of type SP (superfield) with descriptor type D, F or P, and U in column Unique option. The descriptor types have the following meaning:

  - **D**

    Field is a unique index.

  - **F**

    Field is a foreign key with unique index.

  - **P**

    Field is a primary key. This always implies that the field has a unique constraint.

Foreign Keys

Foreign keys are documented as follows:

- If the foreign key consists of only one column, the field documenting the foreign key is marked as a descriptor with descriptor type F or E.

- If the foreign key consists of multiple columns, it is documented as a field with field type SP (superfield) and descriptor type F or E. The descriptor type F means the field is a foreign key and an index. E means the field is a foreign key without an index.

Partitioning Keys

Partitioning keys are documented as field objects in Predict as follows:

If the partitioning key consists of only one column, the field documenting the partitioning key is marked by specification of the partitioning schema in the DBMS extensions

If the partitioning key consists of multiple columns, it is documented as a field with field type SP (superfield) and is marked by specification of the partitioning schema in the DBMS extensions

- If the partitioning key consists of only one column, the field documenting the partitioning key is marked by specification of the partitioning schema in the DBMS extensions.

■ If the partitioning key consists of multiple columns, it is documented as a field with field type SP (superfield) and is marked by specification of the partitioning schema in the DBMS extensions.

Cluster Columns

Cluster columns are documented as field objects in Predict as follows:

■ To define a field as a cluster column, enter Y in the field Cluster column of the corresponding file object.

Column Check Expressions

Check expressions for single columns are documented with verifications of status SQL. The check expression is stored as the rule of the verification.

Check expressions can be edited with the Predict Rule Editor.

Comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

Table Check Expression

A table check expression is a check expression that applies to more than one column. A table check expression is an attribute of a file.

To edit table check expressions, enter Y in the field Check expression of the corresponding file object.

Comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

Naming Conventions for Oracle

Oracle naming conventions must be observed when creating or maintaining Predict objects for Oracle. The following rules apply:

- General Rules
- Constraint Name
- Correlation Name
- Index Name
- Table/View Name
- Column Name

- Delimited Identifier

General Rules

Valid identifiers are from 2 to 32 characters long, must start with an alpha character (A - Z) and may be followed by either an alpha, US national character (#, $, @), a digit or underscore character. Identifiers must comply with these rules if Predict maintenance and generation functions are to be applied.

Concatenated names must not consist of more than 32 characters including the hyphen. See the following paragraphs for details.

Hyphens in names are treated as follows:

- When a table/view is generated from a Predict file object, the hyphen will be transformed into a period (.).
- Because hyphens are used as delimiters, only one hyphen can occur in the file ID.
- The hyphen can be used as a minus sign or negative sign in the field expression or the subselect clause and must then be preceded by a blank.
- Column names must not contain a hyphen.

> **Notes:**

1. Incorporation functions accept object and creator names as selection criteria according to the restrictions mentioned here and in the following sections. Oracle objects with a name longer than the described ones are not offered for incorporation.

2. Names that apply to the restrictions mentioned here and in the following sections but can not be shown in full length are indicated by ">" as last character.

3. For views use the editor commands `FLIP C` and `FLIP T` to edit the field names and their derivation in full length.

Constraint Name

Contraint names can consist of up to 32 characters.

Correlation Name

Correlation names can be defined in the subselect clause of a view. If a correlation name is defined for a table/view in the subselect clause, all references (in field expressions as well as in the field editor of the view) to columns of the table/view must be qualified with the correlation name. If no correlation name is defined for a table/view in the subselect clause, all references to columns of the table/view must be fully qualified with creator-tablename-columnname (for example: SYSORACLE-SYSCOLUMNS-COLNAME).

Correlation names can consist of up to 18 characters.

Index Name

Index names consist of the creator name and the actual name of the index concatenated by a hyphen (used as qualification character).

Creator name and actual name of the index each can consist of up to 32 characters, but a concatenation of both (including the hyphen as qualification character) must not exceed a length of 32 characters.

Table/View Name

Table/View names consist of the creator name and the actual name of the table/view concatenated by a hyphen (used as qualification character).

Creator name and actual name of the table/view each can consist of up to 32 characters, but a concatenation of both (including the hyphen as qualification character) must not exceed a length of 32 characters.

Column Name

Column names can consist of up to 32 characters.

Delimited Identifier

With Oracle, special characters can be used in identifiers of tables and views. Identifiers that contain special characters have to be delimited (usually with single or double quotes) and are therefore called delimited identifiers.

Oracle tables and views with delimited identifiers can be incorporated. They can then be renamed with Predict maintenance functions and retrieval functions can be applied to them. It is strongly advisable to rename delimited identifiers for the following reasons:

■ The only Predict functions that can be applied without restriction to objects with delimited identifiers are Incorporate and Rename.

- If identifiers contain special characters such as blank or asterisk, results of retrieval functions are unpredictable.

- Views can only be generated if the subselect clause and the column expressions do not contain references to delimited identifiers enclosed by quotation marks.

As the SQL escape character Predict uses quotation marks (") and as the SQL string delimiter apostrophes (') are used. The Predict Incorporate function converts other escape characters or string delimiters to quotation marks (") and apostrophes (').

Generating, Incorporating and Comparing Oracle Objects

Prerequisites

Generation, Incorporation and Comparison are subject to Oracle security mechanisms. The following mechanisms apply:

- To perform the Generate function and administration functions Purge and Refresh, the user must have the appropriate privileges within Oracle.

- To perform Incorporation and Comparison functions, the user must have SELECT privilege on nearly all catalog tables.

  - The SELECT privilege in Oracle is the minimum prerequisite for incorporation of Oracle tables/Views (see *GRANT (TABLE or VIEW PRIVILEGES)* in your *Oracle* documentation.

- To use the SQL statements generated by Predict, the corresponding Oracle privileges are also required.

Generation

Oracle objects can be generated from Predict documentation objects.

Oracle table/view	Columns, indexes and referential constraints are automatically included when generating Oracle tables and views. Command: `GENERATE ORACLE-TABLE`

Rules Applying when Generating Oracle Objects

- All objects are generated by first generating the SQL statements that are necessary to implement the object and then executing these statements. An additional confirmation is requested before an Oracle object is actually implemented.

- The generated SQL statements can be saved in a protocol.

- If a generation function is executed for an object that is already implemented, the existing Oracle object can be updated.

See respective sections in *Generation* in the *External Objects in Predict* documentation for more information.

Incorporation

The incorporation functions create Predict documentation objects for tables/views (including columns, indexes and referential constraints) from the system catalog.

See the section *Incorporation* in the *External Objects in Predict* documentation.

Comparison

The comparison functions list differences between the current implementation in Oracle and the corresponding documentation. The documentation can be updated to match the implementation.

See the section *Comparison* in the *External Objects in Predict* documentation.

Administrating Implemented Oracle Objects

Functions for administrating Oracle objects are provided to display, purge or refresh Oracle objects that have been implemented from Predict documentation.

- The Display function lists generation protocols.
- The Purge function drops a table/view physically in Oracle.
- The Refresh function deletes all data in an implemented table but keeps the table structure.

See the section *Administration of External Objects* in the *External Objects in Predict* documentation.

9 Static SQL

With static SQL, data in an SQL-based DBMS (DB2 or SQL/DS) is accessed using an application plan. Accessing data with static SQL is faster than with dynamic SQL.

Natural for DB2 supports the use of static SQL. If a Natural program uses static SQL, a DBRM (database request module) must be generated for that program. In DB2, this DBRM must be included in an application plan.

General Information

Predict supports static SQL in several ways:

- DBRMs can be generated with the Natural for DB2 function CREATE DBRM from information stored in Predict, and Predict documentation of DBRMs can be partially generated from XRef data of existing DBRMs with the program maintenance function Redocument program. Hence Predict supports implementing and documenting static SQL no matter which one of them is done first.

- Programs that use dynamic SQL instead of static SQL can be detected easily.

  - If a program that uses static SQL has been modified and recataloged, the information stored in the DBRM for that program is no longer correct. In this case, the program automatically switches back to the use of dynamic SQL. No action has to be taken by the programmer. The switch back to dynamic SQL is therefore not necessarily recognized by the user (only increasing response times might indicate that dynamic SQL is being used again).

  - Renaming a program that uses static SQL leads to an error at execution time.

- With Predict active retrieval functions, programs that have switched back to the use of dynamic SQL because of modifications or that have been renamed after DBRM generation can be found easily.

Documenting the Use of Static SQL

DBRMs are documented with program objects with language Q (Static SQL). Static SQL is treated in Predict like a Third Generation Language (3GL).

- An implemented DBRM is referenced in a Predict program object by an 8-character member name. A member is a set of XRef data created for the DBRM (as with 3GL programs).

- With members, an 8-character 3GL library name can be specified to identify the load library of the DBRM. A library must have been documented with a Predict system object of type G (3GL Application). This system object can be used to document the load library. If a 3GL application has not yet been documented with a system of type G, the default DBRM library *SYSSTA* is used.

Both the 8-character member name and the 8-character library name belong to the implementation pointer of the program object documenting the DBRM and hence connect the documentation object to the implemented DBRM.

Documenting Which Natural Programs Use a DBRM

A DBRM is typically used by several Natural programs, which may or may not belong to the same library. Programs using the same DBRM must be stored in the same user system file.

For each Natural program using a DBRM, an entry point must be defined in the documentation of the DBRM. The procedure is as follows:

The Natural program is entered in the entry point list with the Link Editor and Predict generates a unique entry point name for this program in the entry point list. Each entry point name is concatenated from

- the DBRM library name (not if the default library *SYSSTA* is used)
- the DBRM member name
- the Natural library name
- the Natural member name.

The Natural-based Link Editor is available for maintaining entry point lists in Predict:

- Natural programs documented in Predict can be selected from a list. See command `SELECT` in the section *Editors in Predict* in the *Predict Reference* documentation.
- If the DBRM has already been implemented and XRef data exists, the entry points can be derived with the commands `ACTIVE` or `UPDATE`.
  - `ACTIVE` reads the entry point names from the XRef data of the DBRM into the editor workspace and marks them as < active. Entry point names that have been entered manually but are not in the XRef data are marked as < unused.
  - `UPDATE` additionally deletes the entry points marked as < unused from the editor workspace.

Generating DBRMs from Predict Documentation

DBRMs can be generated from Predict program objects of language Q with the Natural for DB2 function CREATE DBRM. XRef data for the DBRM can then be created as well.

The names of Natural programs for which DBRMs are to be generated can be specified in two ways:

- Directly as input data to the Natural for DB2 function CREATE DBRM.

- By using the entry point list of the Predict object. In this case, the Predict object to be used for the DBRM creation must be specified with the following options of the CREATE DBRM function:

  - The option USING PREDICT DOCUMENTATION must be specified.

  - The 8-character member name of the DBRM (CREATE DBRM <member name> …).

  - The 3GL library the DBRM is assigned to can be specified with the parameter LIB <library name>. Predict then searches for a DBRM documentation with the <member name> and <library name> in the implementation pointer. If no <library name> is specified, the default library *SYSSTA* is searched for the given <member name>.

In both cases, XRef data can be created (provided that all Natural members themselves have been cataloged with XRef data). The XREF option (N, Y, F) of the Natural for DB2 function CREATE DBRM determines how the generation function behaves with respect to documentation and XRef data for the DBRM:

- If the XREF option is N, no XRef data will be written. Existing XRef data will be deleted.

- If the XREF option is Y, XRef data will be written. Existing XRef data will be overwritten.

- If the XREF option is F, the DBRM generation is only executed if the DBRM has already been documented in Predict. If this is true, XRef data will be written and existing XRef data will be overwritten.

A default value for the XREF option for DBRMs can be defined (use function Defaults > General Defaults > Miscellaneous > Static SQL XREF). This default value can be changed for a single execution of the CREATE DBRM function

- from N to Y or F, or

- from Y to F

It cannot be changed from

- from F to Y or N, or

- from Y to N.

Which Information is Stored in XRef Data

XRef data for a DBRM contains the following information:

- which files and fields are accessed via the DBRM,

- which Natural members use the DBRM. A list of entry names of the DBRM are generated from the names of the Natural members for which the DBRM is generated. The construction of the entry names is the same as in the documentation of the DBRM.

Creation of a DBRM with the XREF option set to Y or F also affects the XRef data of the Natural programs for which the DBRM was created. Since the Natural program now "uses" the DBRM to

access the database with Static SQL, this is noted in the XRef data of the Natural program as a CALL reference to the corresponding entry point in the DBRM (special call-type Static SQL).

Retrieval Functions and Consistency Checking

If a Natural program using static SQL has been modified and recataloged, the DBRM must be regenerated. Otherwise the program will automatically switch back to the use of dynamic SQL. This is reflected in the XRef data written when recataloging the program: It no longer contains a CALL reference to the DBRM.

However, an unused entry point in the documentation of the DBRM remains. This indicates that a Natural program designed for the use of static SQL via a DBRM in fact uses dynamic SQL. It is therefore possible to check if DBRMs have to be regenerated by comparing the Predict program object documenting a DBRM and the corresponding DBRM member (XRef data).

The following functions are available:

- LIST XREF for 3GL functions show XRef data for DBRMs.

- The Verify consistency function of LIST XREF shows all programs which have been renamed after DBRM generation.

- The File Active Retrieval function List files accessed via dynamic SQL shows all DB2 files that are accessed by Natural without using a DBRM.

- The Member Active Retrieval function List members using dynamic SQL shows all Natural members using DB2 files without using a DBRM.

Using Predict Information when Binding Application Plans

Natural for DB2 can use information in Predict to bind plans. See your Natural for DB2 documentation for details. Information in Predict packagelist objects is then used to build the `MEMBER` and the `PKLIST CLAUSE` of a `BIND` statement. The following is created depending on the type of packagelist:

Type of Packagelist	PKLIST CLAUSE or MEMBER Created
Total collection (T)	PKLIST (location.collection.*, ...)
Subcollection (S)	PKLIST (location.collection.package_ID, ...)
Member (Q)	MEMBER (DBRM_name, ...)

> **Note:** The member name of the Predict program object documenting a package is interpreted as the package_ID or DBRM_name.

10 Adabas D and Other SQL Systems

General Information

Predict offers enhanced support for the following SQL systems:

- Adabas D
- Ingres
- Informix
- Sybase

The following SQL objects and attributes can be documented in Predict. Not all attributes are applicable for all SQL systems.

- Tables and Views
- Columns
- Referential constraints
- Unique constraints
- Check constraints
- Stored procedures
- Triggers
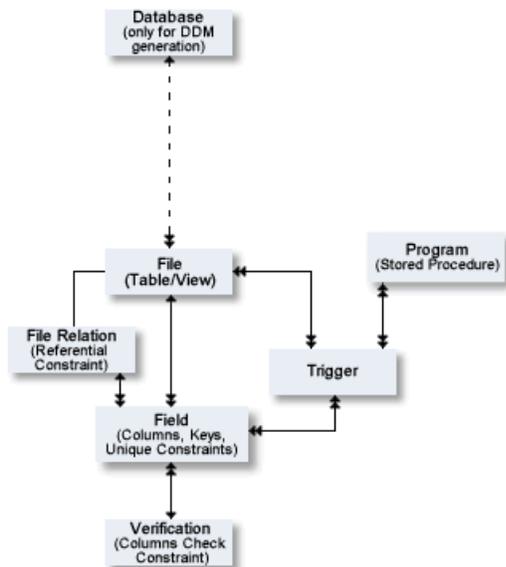
Documenting SQL Systems in Predict

Documenting SQL Tables and Views

The following table gives an overview of how different SQL objects are documented in Predict.

SQL Object	Documented in Predict with File of Type
Adabas D Table, View	BT, BV
Ingres Table, View	JT, JV
Informix Table, View	XT, XV
Sybase Table, View	YT, YV
Other SQL systems	X

Subselect Clauses and Derived Field Expressions

The documentation of views is supported by the Natural-based Subquery Editor in Predict to specify the part of the subselect clause starting from the first `FROM` clause.

The selection clause of the subselect clause is documented by the specification of the field list of the view. The specified list of tables/views in the first `FROM` clause of the subselect clause is generated by Predict and will be updated if a field from an additional table/view is added to the view. Correlation names can be added to the tables and views in the list (using editor functions). The remaining part of the subselect clause is left unchanged.

The expression used to define SQL fields can contain complex expressions. fields that are defined not only by a single column name but use either a constant or a more complex expression are called derived fields. These derived fields can be edited with the Subquery Editor.

Comment lines (starting with * or **) and remarks within a line (starting with /*) are allowed in derived field expressions and subselect clauses.

Documenting Other SQL Objects

SQL Object	Valid for				Documented in Predict with Object of Type	Note
	BT,BV	JT,JV	XT,XV	YT,YV		
Trigger		Y	Y	Y	Trigger	"Triggers" are referred to as "Rules" in Ingres.
Table Check Constraint	Y	Y	Y		Attribute of file	
Column Check Constraint	Y	Y	Y	Y	Verification of status SQL	A "Column check constraint" is referred to as

SQL Object	Valid for				Documented in Predict with Object of Type	Note
	BT,BV	JT,JV	XT,XV	YT,YV		
						"Integrity" in Ingres and "Rule" in Sybase.
Stored Procedure	(Y)	Y	Y	Y	Program of type R	
Column	Y	Y	Y	Y	Field	
Common Key				Y	Field	
Primary Key	Y	Y	Y	Y	Field	
Foreign Key	Y	Y	Y	Y	Field	
Unique constraint	Y	Y	Y	Y	Field	
Referential constraint	Y	Y	Y	Y	File Relation of type R	

Note: Objects marked with (Y) can be documented in Predict but are not included in the respective CREATE statement.

SQL Columns

SQL columns are documented as field objects.

Keys

SQL keys are documented as follows:

- If the key consists of only *one* column, the field documenting the column is marked as a descriptor with descriptor type P, E or K.

- If the key applies to *multiple* columns, it is documented as a field of type SP (superfield) with descriptor type P, E or K. The descriptor types have the following meaning:

  - **E**
    Field is a foreign key.

  - **K**
    Field is a common key.

  - **P**
    Field is a primary key. This always implies that the field has a unique constraint.

Unique Constraints

Unique constraints are documented as follows:

- If the unique constrain applies to only *one* column, the field documenting the column is marked U in column Unique option.

- If the unique constraint applies to *multiple* columns, it is documented as a field of type SP (superfield) with descriptor type P, E or K, and U in column Unique option. The descriptor types have the following meaning:

- **E**

  Field is a unique foreign key.

- **K**

  Field is a unique common key.

- **P**

  Field is a primary key. This always implies that the field has a unique constraint.

Common Keys

Common keys (columns that are frequently joined between two tables or views) are documented in Predict with a file relation of type K. The two fields for which the relationship is to be established must have descriptor type K.

Common keys are only applicable to Sybase.

Referential Constraints

Referential constraints are documented as file relations of type R (referential constraint). A relationship is established between a unique key and a foreign key. Unique and foreign key can belong to the same or to different tables.

Column Check Expressions

Check expressions for single columns are documented with verifications of status SQL. The check expression is stored as the rule of the Verification.

Check expressions can be edited with the Predict Rule Editor.

Comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

Table Check Expressions

A table check expression is a check expression that applies to more than one column. A table check expression is an attribute of a file.

To edit table check expressions, enter Y in the field Trigger of the corresponding file object.

Comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

Triggers

See the section *Trigger* in the *Predefined Object Types in Predict* documentation

Stored Procedure

Stored procedures are documented the procedure code of programs of type R (SQL procedure) and language S (SQL). If the trigger of a file of type XT, YT or JT contains the text EXECUTE pro-

cedure_name, and the procedure_name corresponds to a program of type R and language Q, then the procedure code of the program object is included in the generated `CREATE` statement.

Naming Conventions for SQL Objects

Special naming conventions apply to the following objects in Predict:

- SQL file types. See table below.
- Fields linked as children to these file types
- Constraint names
- Correlation names
- Procedure/Function name

The file IDs must be fully qualified.

A fully qualified ID consists of three parts:

- Creator of up to 8 characters
- Hyphen to separate creator from table/view name
- Table/view name. The maximum length depends on the SQL system. See table below.

Fully qualified IDs may not exceed 32 characters. For SQL objects where the table/view name may not exceed 18 characters, the maximum length of the fully qualified ID in Predict is 27.

The permitted characters listed in the table below apply to creator and table/view name.

Convention	File Type			
	BT, BV	JT, JV	XT, XV	YT, YV
Maximum length of table/view name	18	24	18	30
Upper case				
Upper/lower case	Y	Y	Y	Y
"_" allowed at first position		Y		Y
"#" allowed at first position	Y			
"$" allowed at first position	Y			
"@" allowed at first position	Y			
"_" allowed from second position	Y	Y	Y	Y
"#" allowed from second position	Y	Y		Y
"$" allowed from second position	Y	Y		Y
"@" allowed from sec. position	Y	Y		Y

Convention	File Type			
	BT, BV	JT, JV	XT, XV	YT, YV
Numbers allowed from second position	Y	Y	Y	Y

Correlation Names

Correlation names can be defined in the subselect clause of a view. If a correlation name is defined for a table/view in the subselect clause, all references (in field expressions as well as in the field editor of the view) to columns of the table/view must be qualified with the correlation name. If no correlation name is defined for a table/view in the subselect clause, all references to columns of the table/view must be fully qualified with creator-tablename-columnname (for example: SYSIBM-SYSCOLUMNS-COLNAME).

Delimited Identifiers

It is possible to incorporate SQL tables and views that have delimited identifiers. These tables and views can then be renamed with Predict maintenance functions, and retrieval functions can be applied to them. It is strongly advisable to rename delimited identifiers for the following reasons:

- The only Predict functions that can be applied without restriction to objects with delimited identifiers are Incorporate and Rename.
- If identifiers contain special characters such as blank or asterisk, results of retrieval functions are unpredictable.
- Views can only be generated if the subselect clause and the column expressions do not contain references to delimited identifiers enclosed by quotation marks.

Generating SQL CREATE Statements

Functional Scope

The following table gives an overview of the `CREATE` statements that can be generated from Predict objects with the function Generate SQL CREATE Statement. These SQL statements are stored as Natural members.

If a `CREATE` statement is not available for a particular SQL system, a corresponding clause is generated in the `CREATE TABLE` or `CREATE VIEW` statement if applicable.

CREATE STATEMENT	File Type					
	BT,BV	D,E	JT,JV	X	XT,XV	YT,YV
TABLE, VIEW	Y	Y	Y	Y	Y	Y
INDEX		Y				
DEFAULT					Y	Y
RULE			Y		Y	Y
PROCEDURE			Y		Y	Y
TRIGGER			Y		Y	Y
LABEL ON		Y				
COMMENT ON		Y				

The statements can be punched to an operating system member for further processing, for example execution with an interactive SQL tool or a user program.

More Information

For more information see the section *Generation* in the *External Objects in Predict* documentation.

Generating DDMs from SQL Objects

The following rules apply when generating a DDM for Natural from an SQL file object:

■ The file must be linked via "Contains FI" to a database of a compatible type:

File Type		Compatible Database Type	
BT, BV	Adabas D Table, View	B	Adabas D Handler
JT, JV	Ingres Table, View	J	Ingres Handler
XT, XT	Informix Table, View	X	Informix Handler
YT, YV	Sybase Table, View	Y	Sybase Handler

■ Files documenting both tables and views must be linked to a database of the corresponding type.

■ The file must be linked via "Contains FI" to a database of which the database number is defined in the *NATCONF.CFG* file of type OSQ. For more information see *Data Definition Module* in the section *Generation* in the *External Objects in Predict* documentation.

Incorporating Tables / Views of SQL Database Systems

Incorporation of tables and views of SQL systems is subject to security mechanisms of the respective system.

Tables and views of the following SQL systems can be incorporated.

- Adabas D
- Ingres
- Informix
- Sybase

Required Access

Access is required to the following in the respective SQL system:

Adabas D
- Command SHOW

Ingres
- IICOLUMNS
- IIDBDEPENDS
- IIINTEGRITIES
- IIRELATION
- IITABLES
- IIVIEWS

Informix
- SYSCOLUMNS
- SYSCONSTRAINTS
- SYSDEPEND
- SYSINDEXES
- SYSTABLES
- SYSUSERS
- SYSVIEWS

Sybase
- master.dbo.spt_values
- syscolumns

- syscomments
- sysdepends
- syskeys
- sysobjects
- systypes

More Information

For more information see the section *Incorporation* in the *External Objects in Predict* documentation.

Administrating SQL Objects

You can display and purge the generation protocols created by Predict from the function Generate SQL CREATE statement. You cannot process objects in the external SQL environment with administration functions.

More Information

For more information see the section *Administration of External Objects* in the *External Objects in Predict* documentation.

11 Adabas SQL Server

The Adabas SQL catalog contains all the necessary information on Adabas tables and views. This information can be documented in Predict, and from this documentation a table or view can be created with a Predict generation, incorporation or administration function.

General Information

Predict supports the following SQL statements:

- `CREATE TABLE DESCRIPTION`
- `CREATE CLUSTER DESCRIPTION`
- `CREATE VIEW`
- `DROP TABLE DESCRIPTION`
- `DROP CLUSTER DESCRIPTION`
- `DROP VIEW`

> **Note:** The statements `CREATE TABLE DESCRIPTION` and `CREATE CLUSTER DESCRIPTION` are supported instead of `CREATE TABLE` and `CREATE CLUSTER` to pass existing data structures to the Adabas SQL Server. Also, Predict descriptions already take account of a variety of Adabas-specific features.

Prerequisites

Parts of the ADVANCED Interactive Facilities of Adabas SQL Server must be installed within Natural. For detailed information see the requirements table in the respective part of the *Predict Installation* documentation.

Documenting Adabas SQL Server in Predict

The following Adabas SQL Server objects can be documented in Predict:

- **Adabas tables**
- **Adabas Views**
- **Indexes**
- **Unique elements**
- **Primary and foreign keys**
- **Referential constraints**

Documenting Adabas Tables

There are two methods of documenting Adabas tables:

With Files of Type A - with Adabas SQL usage set to Y

If an Adabas file corresponds *exactly* to a base table in Adabas SQL Server, it can be documented as a file of type A (SQL).

The Adabas file must not contain groups structures or multiple value fields. Rotated fields are not supported with this method.

This method is retained for reasons of compatibility with earlier Predict versions.

With Files of Type AT

Tables can also be documented with files of type AT (Adabas cluster table). Files of this type can be understood as userviews to an Adabas file.

Files of type AT have the following additional attribute:

Table level	0	Only "flat" structures are permitted (no MU or PE fields).
	1	For defining multiple fields and periodic groups.
	2	For defining multiple fields within a periodic group.

There are two methods of documenting periodic groups and multiple value fields in AT files:

- If the occurrences of PE/MU fields are *fixed*, you can use rotated fields in the AT file.

- If the occurrences of PE/MU fields are *variable*, use subtables (AT files at level 1 or level 2).

Documenting Adabas Views

Adabas views can be documented with files of type B.

Subselect Clauses and Expressions in Field Definitions

The documentation of views is supported by an additional editor to specify the part of the subselect clause starting from the first `FROM` clause.

The selection clause of the subselect is documented by the specification of the field list of the view. The specified list of tables/views in the first `FROM` clause of the subselect is generated by Predict and will be updated if a field from an additional table/view is added to the view. Correlation names can be added to the tables and views in the list (using functions of the editor). The remaining part of the subselect clause is left unchanged.

The expression used to define Adabas SQL fields can contain complex expressions. Fields that are defined not only by a single column name but use either a constant or a more complex expression are called derived fields. A special editor is provided for the specification of the expression of derived fields.

In the field expression and in the subselect clause, comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

Documenting Adabas SQL Databases/ Tablespaces

Adabas SQL database and tablespace definitions need not be documented with separate Predict objects:

- Adabas SQL databases need not be documented with separate Predict objects, because the information to which Adabas database an Adabas SQL tablespace belongs is documented with the file-database link.
- The properties of Adabas SQL tablespaces are documented as attributes of Adabas file objects.

The physical implementation of Adabas tables in Adabas can be performed by Predict (with Generate Adabas file or Generate ADACMP/ADAFDU). The description needed by Adabas SQL to address the Adabas files can be generated with the function Generate Adabas table description.

Documenting Adabas SQL Columns

Adabas SQL columns are documented as field objects in Predict.

Adabas SQL Server has fields with data type SEQNO. These fields are documented in Predict with fields of type QN. This data type is used for documenting occurrences of MU or PE fields:

- SEQNO(0) corresponds to the ISN of the underlying Adabas C table
- SEQNO(1) corresponds to the index of a multiple-value or periodic field.
- SEQNO(2) corresponds to a multiple-value field within a periodic group.

For fields of this type, the column Occ represents an individual occurrence of a PE or MU field, and not the maximum number of occurrences.

These fields can be given a name in a table description and can be selected. However, they are only necessary if you want to perform a search operation using individual occurrences. If an AT file contains more than one MU fields or fields from more than one periodic group, it is only possible to address the same occurrence by means of a SEQNO(1) or SEQNO(2) field.

Documenting Indexes

■ The following attributes of indexes can be specified:

  ■ index name,

  ■ definition of index

■ If the index consists of only one column, the field documenting the column is marked as a descriptor with descriptor type D.

■ If the index consists of multiple columns, it is documented as a field with field type SP (superfield) and descriptor type D.

Documenting Unique Elements

Unique elements are documented as Fields with descriptor type D, unique option U and suppression option R.

Documenting Primary and Foreign Keys

Primary Keys

A primary key always includes an index with descriptor option P.

Foreign Keys

Foreign keys are documented as follows:

■ If the foreign key consists of only one column, the field documenting the foreign key is marked as a descriptor with descriptor type F or E.

■ If the foreign key consists of multiple columns, it is documented as a field with field type SP (superfield) and descriptor type F or E. The descriptor type F means the field is a foreign key and an index. E means the field is a foreign key without an index.

Documenting Referential Constraints

Referential constraints are documented as file relations of type R (referential constraint). The relation is established between a unique element and a foreign key. Primary and foreign key must belong to different subtables. The subtables themselves must belong to the same Adabas file.

Naming Conventions for Adabas SQL Server

Adabas SQL naming conventions have to be followed when creating or maintaining Predict objects for Adabas SQL. The following rules apply:

- Valid identifiers are from 2 to 32 characters long, must start with an alpha character (A - Z) and may be followed by either an alpha, a digit or underscore. Identifiers must obey these rules if Predict maintenance and generation functions are to be applied.

- The identifier of a table or view must be given in qualified form: the schema identifier (maximum length 32 characters), a delimiter and the table/view name (maximum length 32 characters). A hyphen is used as a delimiter (not a period as in SQL). An example: SYSSAG-SYSCOLUMNS. Hyphens in names are treated as follows:

  - When a table/view is generated from a Predict table/view object the hyphen will be transformed into a period (.).

  - Because hyphens are used as delimiters, only one hyphen can occur in the SQL identifier. Column names must not contain a hyphen.

  - The hyphen can be used as a minus sign or negative sign in the field expression or the subselect clause and must then be preceded by a blank.

Correlation Names

Correlation names can be defined in the subselect clause of a view. If a correlation name is defined for a table/view in the subselect clause, all references (in field expressions as well as in the field editor of the view) to columns of the table/view must be qualified with the correlation name. If no correlation name is defined for a table/view in the subselect clause, all references to columns of the table/view must be fully qualified with creator-tablename-columnname (for example: SYS-SAG-SYSCOLUMNS-COLNAME).

Index Names

Index names must be fully qualified: schema, delimiter, index name.

Generating, Incorporating and Comparing Adabas SQL Objects

Prerequisites

The following Predict functions are subject to SQL security mechanisms:

- Function Generate Adabas Table/View and administration functions Purge and Refresh: When the catalog is accessed for the first time, the user ID DBA is used for read access to the catalog.

- Generate Adabas Table/View: If a description is generated into a schema which is not owned by the catalog user, a window appears in which you can enter the ID and password of the schema owner. In batch mode, use the command `SET SCHEMA_OWNER`.

- Incorporation and Comparison functions: User must have SELECT privilege for the schema definition_schema which is delivered with Adabas SQL Server.

Generate Table Description, Cluster Description

With the Predict function Generate Adabas Table/View, a `CREATE TABLE DESCRIPTION` statement or a `CREATE CLUSTER DESCRIPTION` statement is generated from a file of type A (with SQL usage set to Y) or from a file of type AT. See table below:

Constellation	Generated Command	Note
One Adabas file corresponding to one SQL table	`GENERATE TABLE DESCRIPTION`	
One AT file corresponding to one SQL table		
Multiple AT files corresponding to multiple SQL tables	`GENERATE CLUSTER DESCRIPTION`	Specify only one file in the cluster. All files in the cluster will be used for generation.

The statements add descriptions of multiple tables in an Adabas SQL catalog *without* creating an Adabas file (the standard SQL statements `CREATE TABLE/CLUSTER` generate both an Adabas file and a description in an Adabas SQL catalog).

This description contains the following:

- a list of fields in the file

- details of descriptors and superdescriptors, unique constraints, primary and foreign keys

- database name and file number. This information is used to access the Adabas file.

  The database name is taken from the catalog. If no database name exists, a `CREATE DATABASE` statement is generated. If the ID of the Predict database object complies with SQL naming conventions, this name is taken. If not, the database name DB_DBnr is generated.

The available options are described under *Adabas Table/Cluster/View* in the section *Generation* in the *External Objects in Predict* documentation.

If the database/file number is a logical file number or the number of an anchor file (with a file of type Expanded), the table description is appended with "MODIFICATION NOT ALLOWED". This has the result that `ALTER` statements for this table are rejected. Only DML (Data Manipulation Language) statements can be executed for tables marked in this manner; DDL (Data Definition Language) statements are not possible except `DROP TABLE DESCRIPTION`.

The generated SQL statements can be saved in a protocol.

Generate View

With the Predict function Generate Adabas Table/View, a `CREATE VIEW` statement is generated from a file of type B.

The available options are described under *Adabas Table/Cluster/View* in the section *Generation* in the *External Objects in Predict* documentation.

Incorporate Table

With the Predict function Incorporate Adabas Table/View, a file of type A (with SQL usage set to Y) or a file of type AT is documented in Predict from the entry in the Adabas SQL catalog. The constellation in the catalog determines which file type is incorporated in Predict:

Constellation	Incorporated Object
Either ■ External object is contained in the catalog as a table and created with `CREATE TABLE / CREATE TABLE DESCRIPTION`, or ■ Cluster with one table and no rotating fields.	File of type A (with SQL usage) set to Y.
Master Adabas file exists in Predict (determined using DBnr/Fnr).	File of type AT.

The database number is interpreted as a physical Adabas database and a link is created from this database to the Adabas file. The values of the Adabas attributes can either be taken from the tablespace attributes in the catalog, or default values are used which can be adapted to the "real world" with the function Compare Adabas File.

Incorporate View

With the Predict function Incorporate Adabas Table/View, a file of type B is documented in Predict from the entry in the Adabas SQL catalog.

Compare Adabas Table/View

Files of type A (SQL), type AT and type B are compared with information contained in the description of the table/view in an Adabas SQL catalog. See the section *Comparison* in the *External Objects in Predict* documentation for more details.

Administrating Adabas SQL Server Objects

The following administration functions are valid for Adabas table descriptions. Enter function code L and object code FI in any Predict main menu or the command `ADMINISTRATE FILE`. File type is A (SQL), AT or B and external object code is EQ.

For more information see the section *Files* in the section *Administration of External Objects* in the *External Objects in Predict* documentation.

Disconnect Implementation

Deletes the generation protocol and the generation pointer from the Predict File object to the Adabas table description, but the table description is left intact.

With files of type AT, all files used for generation are disconnected in a single operation.

Display Implementation

Displays documentation data, generation options and generated table description of specified file ID(s).

Rename Implementation

Moves the generation protocol to another member and/or library.

Purge Implementation

Deletes Adabas table descriptions and all dependent views.

A `DROP TABLE DESCRIPTION`, `DROP CLUSTER DESCRIPTION` or `DROP VIEW` statement will delete the definition from the Adabas SQL catalog and any statements referencing this table / view are marked as invalid.

When a table / view is dropped, all dependent views are dropped too.

> **Note:** The function is equivalent to the function Generate with Replace table/view set to Y.

Select Implementation

Selects Adabas table/view for further processing.

XRef Data for Adabas SQL Server Objects

Programs using embedded SQL must be precompiled with the Adabas SQL precompiler before the host language compiler is executed.

During precompilation, an option is available to create XRef data. The creation of XRef data is controlled by options specified in the Adabas SQL parameters. See the Adabas SQL documentation for a detailed description of these options.

As described in the section *Third Generation Languages* in this documentation, XRef data is always assigned to members contained in logical libraries. The library containing the member can be specified explicitly or - if no library is specified - the member is assigned to a default library depending on the host language. See table below.

Host Language	Default-Library
C	*SYSCCC*
COBOL	*SYSCOB*
PL/I	*SYSPLI*
FORTRAN	*SYSFOR*

The following XRef data is stored for programs processed by the Adabas SQL precompiler:

- directory information (user ID, terminal ID, date and time of precompilation)
- each Copy/Include Code member used in the program
- each table name used in certain clauses of DML statements, together with the usage:

Statement	Clause	Usage
SELECT	FROM	Read
INSERT	INTO	Store
UPDATE		Update
DELETE	FROM	Delete

■ each column name used in certain clauses of DML statements, together with the usage:

Statement	Clause	Usage
SELECT		Read
SELECT	WHERE	Search
INSERT	INTO	Store
UPDATE	<row amendment expression>	Update
UPDATE	WHERE	Search
DELETE	WHERE	Search

12   Adabas SQL Gateway

The ConnX Data Dictionary (CDD) contains all the necessary information on Adabas tables and views. This information can be documented in Predict, and from this documentation a table or view can be created with a Predict generation, incorporation or administration function.

General Information

Predict supports the following SQL statements:

- `CREATE TABLE DESCRIPTION`
- `CREATE CLUSTER DESCRIPTION`
- `CREATE VIEW`
- `DROP TABLE DESCRIPTION`
- `DROP CLUSTER DESCRIPTION`
- `DROP VIEW`

> **Note:** The statements `CREATE TABLE DESCRIPTION` and `CREATE CLUSTER DESCRIPTION` are supported instead of `CREATE TABLE` and `CREATE CLUSTER` to pass existing data structures to the Adabas SQL Gateway. Also, Predict descriptions already take account of a variety of Adabas-specific features.

Prerequisites

Natural SQL Gateway must be installed within Natural. For detailed information see the requirements table in the respective part of the *Predict Release Notes* documentation.

When a connection to a server is established, a user ID and password may be required. The user ID and password can be changed within a Predict session by executing the command SET CATALOG_USER. This allows you to overwrite the previously entered user ID and password. This user ID and the password are then also used for every other connection to an SQL server.

Documenting Adabas SQL Gateway in Predict

The following Adabas SQL Gateway objects can be documented in Predict:

- **Adabas Tables**
- **Adabas Views**
- **Indexes**
- **Unique elements**
- **Primary and foreign keys**

■ **Referential constraints**

Documenting Adabas Tables

There are two methods of documenting Adabas tables:

With Files of Type A - with Adabas SQL usage set to Y

If an Adabas file corresponds *exactly* to a base table in Adabas SQL Gateway, it can be documented as a file of type A (SQL).

The Adabas file must not contain groups structures or multiple value fields. Rotated fields are not supported with this method.

This method is retained for reasons of compatibility with earlier Predict versions.

With Files of Type AT

Tables can also be documented with files of type AT (Adabas cluster table). Files of this type can be understood as userviews to an Adabas file.

Files of type AT have the following additional attribute:

Table level	0	Only "flat" structures are permitted (no MU or PE fields).
	1	For defining multiple fields and periodic groups.
	2	For defining multiple fields within a periodic group.

There are two methods of documenting periodic groups and multiple value fields in AT files:

■ If the occurrences of PE/MU fields are *fixed*, you can use rotated fields in the AT file.

■ If the occurrences of PE/MU fields are *variable*, use subtables (AT files at level 1 or level 2).

Documenting Adabas Views

Adabas views can be documented with files of type B or CX.

In files of type B only Adabas tables or views can be referenced. In files of type CX any SQL file can be referenced.

Subselect Clauses and Expressions in Field Definitions

The documentation of views is supported by an additional editor to specify the part of the subselect clause starting from the first `FROM` clause.

The selection clause of the subselect is documented by the specification of the field list of the view. The specified list of tables/views in the first `FROM` clause of the subselect is generated by Predict and will be updated if a field from an additional table/view is added to the view. Correlation names can be added to the tables and views in the list (using functions of the editor). The remaining part of the subselect clause is left unchanged.

The expression used to define Adabas SQL fields can contain complex expressions. Fields that are defined not only by a single column name but use either a constant or a more complex expression are called derived fields. A special editor is provided for the specification of the expression of derived fields.

In the field expression and in the subselect clause, comment lines (lines starting with * or **) and remarks within a line (starting with /*) are allowed.

Documenting Adabas SQL Columns

Adabas SQL columns are documented as field objects in Predict.

Adabas SQL Gateway has fields with data type SEQNO. These fields are documented in Predict with fields of type QN. This data type is used for documenting occurrences of MU or PE fields:

- SEQNO(0) corresponds to the ISN of the underlying Adabas C table
- SEQNO(1) corresponds to the index of a multiple-value or periodic field.
- SEQNO(2) corresponds to a multiple-value field within a periodic group.

For fields of this type, the column Occ represents an individual occurrence of a PE or MU field, and not the maximum number of occurrences.

These fields can be given a name in a table description and can be selected. However, they are only necessary if you want to perform a search operation using individual occurrences. If an AT file contains more than one MU fields or fields from more than one periodic group, it is only possible to address the same occurrence by means of a SEQNO(1) or SEQNO(2) field.

Documenting Indexes

- The following attributes of indexes can be specified:
  - index name,
  - definition of index
- If the index consists of only one column, the field documenting the column is marked as a descriptor with descriptor type D.
- If the index consists of multiple columns, it is documented as a field with field type SP (superfield) and descriptor type D.

Documenting Unique Elements

Unique elements are documented as Fields with descriptor type D, unique option U and suppression option R.

Documenting Primary and Foreign Keys

Primary Keys

A primary key always includes an index with descriptor option P.

Foreign Keys

Foreign keys are documented as follows:

- If the foreign key consists of only one column, the field documenting the foreign key is marked as a descriptor with descriptor type F or E.

- If the foreign key consists of multiple columns, it is documented as a field with field type SP (superfield) and descriptor type F or E. The descriptor type F means the field is a foreign key and an index. E means the field is a foreign key without an index.

Documenting Referential Constraints

Referential constraints are documented as file relations of type R (referential constraint). The relation is established between a unique element and a foreign key. Primary and foreign key must belong to different subtables. The subtables themselves must belong to the same Adabas file.

Naming Conventions for Adabas SQL Gateway

Adabas SQL naming conventions have to be followed when creating or maintaining Predict objects for Adabas SQL. The following rules apply:

- Valid identifiers are from 2 to 32 characters long, must start with an alpha character (A - Z) and may be followed by either an alpha, a digit or underscore. Identifiers must obey these rules if Predict maintenance and generation functions are to be applied.

- The identifier of a table or view must be given in qualified form: the schema identifier (maximum length 32 characters), a delimiter and the table/view name (maximum length 32 characters). A hyphen is used as a delimiter (not a period as in SQL). An example: SYSSAG-SYSCOLUMNS. Hyphens in names are treated as follows:

    - When a table/view is generated from a Predict table/view object the hyphen will be transformed into a period (.).

    - Because hyphens are used as delimiters, only one hyphen can occur in the SQL identifier. Column names must not contain a hyphen.

- The hyphen can be used as a minus sign or negative sign in the field expression or the subselect clause and must then be preceded by a blank.

Correlation Names

Correlation names can be defined in the subselect clause of a view. If a correlation name is defined for a table/view in the subselect clause, all references (in field expressions as well as in the field editor of the view) to columns of the table/view must be qualified with the correlation name. If no correlation name is defined for a table/view in the subselect clause, all references to columns of the table/view must be fully qualified with creator-tablename-columnname (for example: SYS-SAG-SYSCOLUMNS-COLNAME).

Index Names

Index names must be fully qualified: schema, delimiter, index name.

Creating Adabas Cluster Tables Using Program CXXGTD

The program `CXXGTD` creates Adabas cluster tables for a given Adabas File.

`CXXGTD` is located in library SYSDIC. When started, you have to specify the name of the documented Adabas file for which you want to define the cluster tables. This file has to be documented in Predict.

The program uses 2 workfiles (each having LRECL=80):

- Work file 1 contains the directives that control the layout of the resulting structure.

  The directives are described below in **Input File Language Syntax**.

  > **Note:** The directives `SYSTRANS`, `DATABASE`, `PRIMARY` and `FOREIGN` are ignored.

  Primary and Foreign keys will always be built using `SEQNO` columns. Work file 1 must contain at least a `SCHEMA` directive.

- Work file 2 will contain the resulting structure in a syntax that can be loaded into Predict using the program `CXXEX`. After a successful load, use the Predict function Generate SQL Create Statements to generate the `CREATE TABLE/CLUSTER DESCRIPTION` statements.

Input File Language Syntax

The input file is used to provide information to program `CXXGTD`. All aspects of the generation can be controlled using the provision of an input file. The file consists of directives. Each directive is started by a new line and is terminated by an end of line. The order of the directives is of no significance. The program `CXXGTD` is only able to process one Adabas file at a time.

Comments

Comments are delimited by the appearance of a # character in the first column and by a new line.

Schema Identifier

In order to specify the schema identifier, the following directive is used:

`SCHEMA` *`schema_name`*

For example:

```
SCHEMA production_schema
```

File Specification

The source Adabas file is specified by the following syntax:

`FILE` *`file_nr`* `[DDM=`*`ddm_name`*`] [[`*`cluster_name`*`]` *`table_name`*`]`

Where:

file_nr	specifies the Adabas file from which the description statement will be generated.
table_name	optionally specifies the table name which will be used either for the master table or for the single table if there is no cluster.
cluster_name	optionally specifies the cluster name which will be used in the generated statement. If both a cluster name and a table name are present, then this forces the generation of a `CREATE CLUSTER DESCRIPTION` statement, even if such a statement is not strictly necessary.

For example:

```
FILE 32 cluster_employees employees
```

Subtable Specification

A subtable can be built around a PE group, a single MU field or a number of MU fields which act in parallel. The subtable directive is able to express either of these possibilities and to enable the optional specification of an identifier for the resulting subtable. In the first two cases, only a single field is required. In the third case, numerous MU fields can be specified in the form of a list separated by commas. The syntax is as follows:

```
short_name [, short_name]... SUBTABLE [table_name]
```

Where:

short_name	is as it suggests.
table_name	represents the intended table identifier for the subtable.

For example:

```
AB SUBTABLE
```

Would map the field AB to a subtable and would generate a table name accordingly.

```
AC SUBTABLE Employee_number
```

Would map the field AC to a subtable but would use the name as indicated.

```
AE, AF SUBTABLE Employee_status
```

Would map the two MU fields AE and AF to the single subtable Employee_status.

Rotated Field Interpretation

If a field is to be interpreted as a rotated field, then the following syntax applies:

```
short_name ROTATE value
```

Where:

short_name	is as it suggests.
value	is the fixed number of occurrences to be rotated e.g. 12 in our example above.

For example:

```
MA ROTATE 12
```

Would map the twelve occurrences of the field MA to twelve individual SQL columns.

Longalpha Field Interpretation

If an MU field of type character is to be interpreted as a longalpha field, then the following syntax applies:

```
short_name LONGALPHA value
```

Where:

short_name	is as it suggests.
value	is the number of bytes which are to be considered in the character field.

For example:

```
LT LONGALPHA 512
```

Would map the field `LT` to a character column of 512 characters.

Long Name Specification

A column identifier for a particular field can be specified as follows:

```
NAME short_name [index] column_name
```

Where:

short_name	is as it suggests.
index	optionally refers to the occurrence number of an MU if the field is rotated.
column_name	is a valid SQL identifier representing the intended column identifier.

For example:

```
NAME MA 3 BONUS_MARCH
```

Field Suppression

If a field is not to be included in a description, then its suppression is specified as follows:

```
SUPPRESS short_name
```

Where:

short_name	is as it suggests.

For example:

```
SUPPRESS BA
```

Data Type Generation

In general, program `CXXGTD` does not generate the explicit data type declaration for a column in the resulting generated statement. The explicit data type declaration is however not strictly necessary as Adabas SQL Gateway can operate on a minimum create table/cluster statement, i.e. the server will fill in any missing information automatically. In such a case, the resulting generated

---

statement can be significantly longer. The forced generation of the column's data type is helpful for documentation purposes. The syntax is as follows:

`DATATYPE`

Error Handling

Should program `CXXGTD` encounter an error condition during execution, then it will generally terminate with an exit code and in addition an error message. The possible exit codes are:

0	Successful completion.
4	Warning generated.
8	Error detected.

The error text can be found in `SYSOUT`.

Generating, Incorporating and Comparing Adabas SQL Objects

Generate Table Description, Cluster Description

With the Predict function Generate Connx Dictionary , a `CREATE TABLE DESCRIPTION` statement or a `CREATE CLUSTER DESCRIPTION` statement is generated from a file of type A (with SQL usage set to Y) or from a file of type AT. See table below:

Constellation	Generated Command	Note
One Adabas file corresponding to one SQL table	`GENERATE TABLE DESCRIPTION`	
One AT file corresponding to one SQL table		
Multiple AT files corresponding to multiple SQL tables	`GENERATE CLUSTER DESCRIPTION`	Specify only one file in the cluster. All files in the cluster will be used for generation.

The statements add descriptions of multiple tables in an Adabas SQL catalog *without* creating an Adabas file (the standard SQL statements `CREATE TABLE/CLUSTER` generate both an Adabas file and a description in an Adabas SQL catalog).

This description contains the following:

- a list of fields in the file

- details of descriptors and superdescriptors, unique constraints, primary and foreign keys

- database name and file number. This information is used to access the Adabas file.

The available options are described under *Adabas Table/Cluster/View* in the section *Generation* in the *External Objects in Predict* documentation.

The generated SQL statements can be saved in a protocol.

Generate View

With the Predict function Generate Connx Dictionary, a `CREATE VIEW` statement is generated from a file of type B or CX.

The available options are described under *Adabas Table/Cluster/View* in the section *Generation* in the *External Objects in Predict* documentation.

Incorporate Table

With the Predict function Incorporate Connx Dictionary, a file of type A (with SQL usage set to Y) or a file of type AT is documented in Predict from the entry in the Adabas SQL catalog. The constellation in the catalog determines which file type is incorporated in Predict:

Constellation	Incorporated Object
Base table	File of type AT.
View	File of type CX.

Incorporate View

With the Predict function Incorporate Connx Dictionary, a file of type B or CX is documented in Predict from the entry in the Adabas SQL catalog.

Administrating Adabas SQL Gateway Objects

The following administration functions are valid for Adabas table descriptions. Enter function code L and object code FI in any Predict main menu or the command `ADMINISTRATE FILE`. File type is A (SQL), AT, B or CX and external object code is ZD.

For more information see the section *Files* in the section *Administration of External Objects* in the *External Objects in Predict* documentation.

Disconnect Implementation

Deletes the generation protocol and the generation pointer from the Predict File object to the Adabas table description, but the table description is left intact.

With files of type AT, all files used for generation are disconnected in a single operation.

Display Implementation

Displays documentation data, generation options and generated table description of specified file ID(s).

Rename Implementation

Moves the generation protocol to another member and/or library.

Purge Implementation

Deletes Adabas table descriptions and all dependent views.

A `DROP TABLE DESCRIPTION`, `DROP CLUSTER DESCRIPTION` or `DROP VIEW` statement will delete the definition from the Adabas SQL catalog and any statements referencing this table / view are marked as invalid.

When a table / view is dropped, all dependent views are dropped too.

> **Note:** The function is equivalent to the function Generate with Replace table/view set to Y.

Select Implementation

Selects Adabas table/view for further processing.

13 **Third Generation Languages**

Predict provides functions for documentation, development and redocumentation of 3GL applications and programs. The following third generation languages are supported:

- BAL/Assembler
- C
- COBOL
- FORTRAN
- PL/I
- Ada

Not all facilities are available for all of these languages. On the other hand there are areas in Predict where additional user-specified languages are supported. In other areas, special classes of programs are used that are treated by Predict like languages. These restrictions and extensions are either mentioned here or in the relevant section of this documentation or the *External Objects in Predict* documentation.

Documenting 3GL Applications

3GL applications are documented in Predict with system objects of type 3GL Application (code G).

The system object and the 3GL application it represents are connected by an implementation pointer.

Implementation Pointer for 3GL Application

Parameters	
Library	This name can be freely chosen when an object of type system is added or modified, and represents one or more source or load libraries/directories or parts thereof.
	Once defined here, this name can be used to document 3GL programs belonging to the application and for creating and retrieving XRef data.
	If XRef data related to this library exists, the name may no longer be changed.
User system Fnr, DBnr	These attributes are used to distinguish 3GL libraries from Natural libraries. Both must be set to 255 for 3GL applications.

Documenting a 3GL Application with a Predict Object of Type System
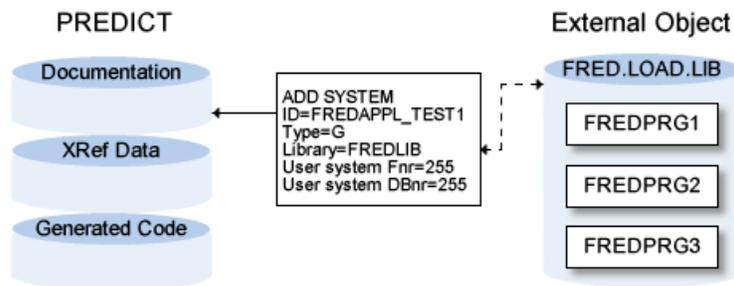
System implementation pointer Library represents for Predict one or more source or load librar-ies/directories or parts thereof.

Documenting 3GL Programs

3GL programs are documented in Predict with objects of type Program, with one of the languages listed on page 2 and one of the following subtypes, depending on the programming language.

- copy code
- documented
- program
- function
- subprogram.

Other languages can be defined in the user exit U-PGMLAN. See the section *User Exits* in the *Predict Administration* documentation for more information.

Predict also knows the pseudo-languages **System program** and **Static SQL**.

As with applications, the program object and the implemented 3GL member it represents are connected by an implementation pointer.

Implementation Pointer for 3GL Programs

If Member is not entered for the program implementation pointer, it is not possible to enter a value for Library. If a Member is specified, the possible values for Library depend on how the maintenance option Implementation library described below has been defined by your DDA in the General Defaults function.

Presetting		
Implementation library	F	Force. A library that is documented as a 3GL application must be entered. A default library - for example *SYSCOB* - may not be entered.
	A	Allowed. Either a library documented as a 3GL application or a default library must be entered. See next table for a complete list of default libraries for 3GL programs.
	D	Disallowed. Library concept is not used. Library *SYSALL* must be entered.

Parameters	
Member	Corresponds to the name of the implemented program as it is stored in a source or load library/directory.
Library	The possible values for this parameter depend on the maintenance option Implementation Library. See *Presetting* above.
	Corresponds to the implemented application to which the program belongs. If a non-default library is specified, it must be defined in an object of type system if XRef data is to be created for the program.
	The program object does not have to be linked in Predict to the system object containing the library name. If Member is specified but Library is left blank, and if Implementation Library is set to A, the program object is connected automatically to the corresponding default library:

Language	Default Library
COBOL	*SYSCOB*
BAL/Assembler	*SYSBAL*
PL/I	*SYSPLI*
FORTRAN	*SYSFOR*
C	*SYSCCC*
ADA	*SYSADA*

Parameters		
	Pseudo-Language	**Default Library**
	Static SQL	*SYSSTA*
	System program	*SYSSYS*
	Default libraries do not need to be defined explicitly in a system object. **Note:** Programs of language System program *must* be linked to library *SYSSYS*. Programs of all other languages in the above list can be linked either to their default library or to a user-defined library defined in a system object.	
User system Fnr, DBnr	These attributes are used to distinguish implemented 3GL programs from Natural programs. Both must be set to 255 for 3GL programs.	

Documenting a 3GL Program with a Predict Object of Type Program
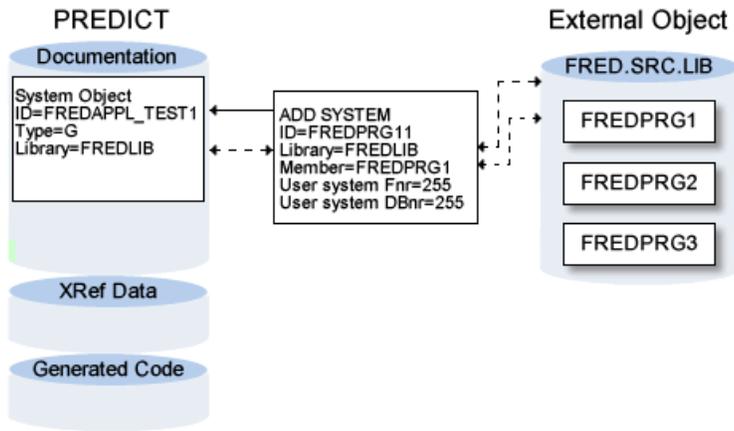
Program implementation pointer Member corresponds to the implemented program as it is stored in the source or load library/directory. Library must be defined in an object of type System if XRef data is to be created for the program.

Creating XRef Data for Implemented Programs
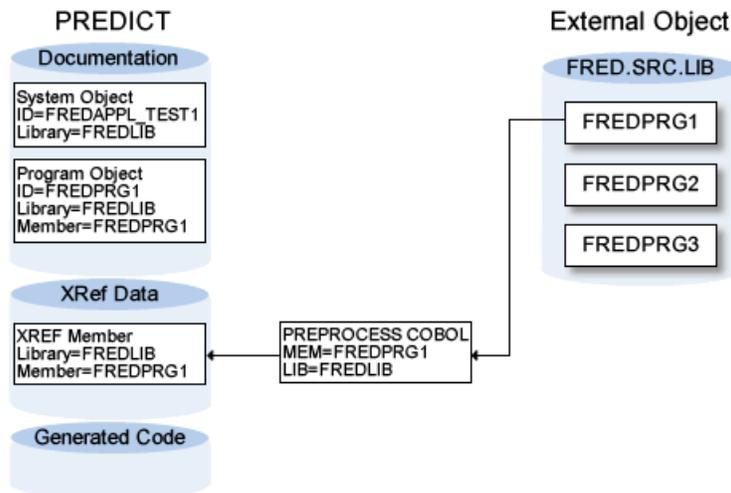
An implemented program is known to Predict only if XRef data exist. In this example, the XRef data for COBOL program FREDPRG1 are created by assigning the program source as Workfile 1 and then processing it with the Predict Preprocessor.

The different methods of creating XRef data are listed in the section *XRef Data for 3GL Applications and Programs*.

Connecting External and Documentation Objects by Implementation Pointer
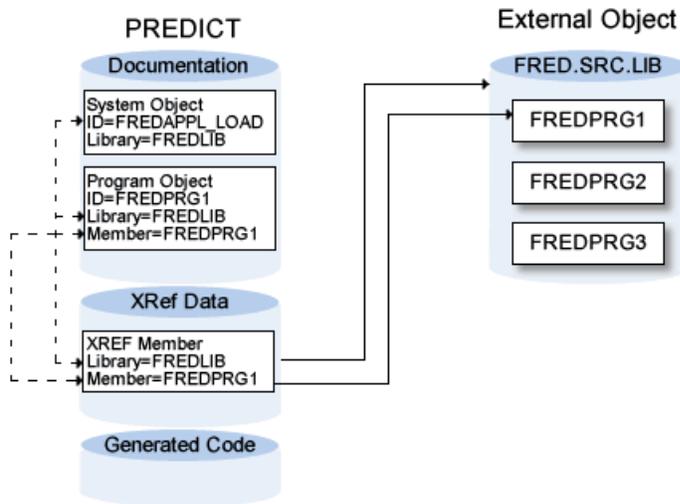
The Predict system object is now connected to external library, and the program object is now connected to the implemented program.

Documenting Entry Points for 3GL Programs

Entry points can be documented in Predict for the following languages:

- Assembler
- C
- COBOL
- FORTRAN
- PL/I
- Ada
- Other (language code O)

If a member name is entered in the implementation pointer of a program object, this name is automatically entered as an entry point. Other entry points can be entered using one of the following methods:

- **Documenting entry points manually**
  - with program maintenance function Edit entry-points (Code R)
  - or by setting Attribute to Y, and then selecting Entry points in the Additional attributes window.

- **Documenting entry points with editor commands**
  If XRef data already exists for the implemented program, call the function Edit Entry Points with one of the methods above and enter command `ACTIVE` or `UPDATE` in the editor command line.

  - `ACTIVE` reads the entry names from the XRef data of the program into the editor workspace and marks them as < active. Entry names that have been entered manually but are not in the XRef data are marked < unused.

  - `UPDATE` additionally deletes the entries marked < unused from the editor workspace.

  See *Program List Specific Editor Commands* in the section *Program* in the *Predefined Object Types in Predict* documentation.

- **Documenting entry points with function Redocument program**
  See *Redocumenting of 3GL Applications* or *Redocument Program* section *Program* in the *Predefined Object Types in Predict* documentation for more information.

XRef Data for 3GL Applications and Programs

XRef data for 3GL programs plays two important roles in Predict:

- It contains information on the dependencies among implemented programs and between programs and other objects they use.

- It represents the implemented program in Predict. This means an XRef member corresponding to the implemented program must exist if the program is to be known to Predict.

How is XRef Data Created?

XRef data for applications is created by creating XRef data for one or more programs contained within the application. The method used for creating XRef data for programs depends on the program type:

For 3GL Programs

- By Adabas Native SQL (ADA, COBOL, FORTRAN and PL/I).

- By the Predict Preprocessor (Assembler, COBOL and PL/I). See the section *Preprocessor* in the *External Objects in Predict* documentation.

- By Adabas SQL Server precompiler (C, COBOL, FORTRAN and PL/I). See *XRef Data for Adabas SQL Objects*.

For Static SQL

If Natural for DB2 is installed, the function CREATE DBRM of Natural for DB2 creates XRef data for Static SQL access modules (DBRMs) and for Natural programs that use Static SQL. See *Static SQL*.

For System Programs

It is not always possible to create XRef data for a 3GL program using one of the above methods. This applies particularly to operating system routines, TP Monitor programming interfaces or other programs that are invoked from within a 3GL application but for which no source code is available. However, these programs can be documented as program objects of type Documented or External program (subtypes D or E) with pseudo-language System program (language code Z).

For each program object of this type, Predict creates a minimal set of XRef data, containing directory information and a list of entry points.

What is Contained in 3GL XRef Data?

The following information is stored for 3GL programs:

- The name of the program and the application to which it belongs.
- The program type (only main program).
- The date and time the program was last cataloged.
- ID of the user who cataloged the program.
- ID of the terminal from which the program was cataloged. In batch mode the job name is given.
- The entry points defined in the program. The member name is always entered as one of the entry points.
- The entry points of invoked 3GL programs and the methods used to invoke them (only CALL, static SQL).
- The names of files used in the program and the type of file usage.
- Names of fields of files used in the program and the type of field usage.

How is XRef Data Used?

There are three main areas where XRef data is used:

- **Active Retrieval**
  Predict active retrieval functions evaluate XRef data and Predict documentation data to determine

  - if objects documented in the dictionary are not yet implemented

  - if implemented programs are not yet documented or

  - if documentation data differs from the implementation.

  XRef data also provides answers to questions such as

  - which programs refer to file *ABC**

  - which programs call the entry point MAIN in program START in library FREDLIB.

  For more information see the section *Active Retrieval* in the *Predict Reference* documentation.

- **LIST XREF**
  XRef data for third generation languages is retrieved with functions of the Predict XRef menu. There are essentially three groups of functions: those which

  - retrieve information on specific types of objects in an application

  - retrieve information on the consistency of an application as a whole

  - manage sets.

  For more information see the section *LIST XREF for Third Generation Languages* in the *Predict Reference* documentation.

- **Redocumenting of 3GL Applications**
  3GL applications for which XRef data exists can be redocumented automatically in Predict. See *Redocumenting of 3GL Applications*.

Using Predict Functions When Developing 3GL Applications

Two major features are available for the development of 3GL applications:

- Generation of file layouts from Predict file objects in the syntax of several third generation languages. See appropriate sections in the section *Generation* in the *External Objects in Predict* documentation.

- Insertion of Predict generated file layouts and Adabas format buffers into 3GL source programs by the Predict Preprocessor. See the section *Preprocessor* in the *External Objects in Predict* documentation for more information.

Redocumenting of 3GL Applications

3GL applications for which XRef data exists can be automatically redocumented in Predict. The XRef data must have been created using one of the methods described in *XRef Data for 3GL Applications and Programs*.

The Redocument program function (see the section *Program* in the *Predefined Object Types in Predict* documentation) creates for each implemented program a new Predict Program object or updates an existing object and evaluates the XRef data to establish links to other program and file objects.

This results in a basic documentation of the application objects and their relationships, which can be extended by an abstract, extended description, keywords, owners etc.

Redocumenting COBOL Record Structures

Data definitions in the form of COBOL Copy Code members can be redocumented in Predict using the function Incorporate COBOL. A file object of type Sequential is created for each Copy Code member. See appropriate part of section *Incorporation* in the *External Objects in Predict* documentation for more information.

14   Predict and Natural Development Server

Natural Development Server (NDV) stores the following information into Predict:

- Structure of Application Descriptions (APD)
- Locks

📄 **Notes:**

1. Locks are internal used objects only and can not be maintained by Predict.
2. For more information about Applications see *Introducing Natural Single Point of Development*.

Documenting Natural Development Server in Predict

The following NDV objects can be documented in Predict:

- Base Applications Descriptions
- Compound Applications Descriptions
- Data Definition Modules
- Natural Programming Objects
- Libraries
- Message files
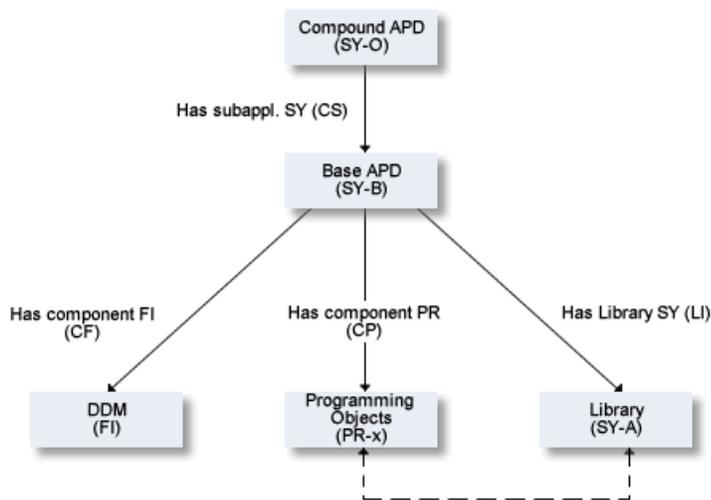
The following table provides an overview of how different NDV objects are documented.

Natural Development Server	Documented in Predict with
Base APD	System object of type B
Compound APD	System object of type O
DDM	File object
Natural Programming Objects	Program object of corresponding type
Library	System object of type A
Message file	Program object of type 2

Documenting Base Application Descriptions

Base Application Descriptions are documented as objects of type System with system type B. Base Application Descriptions have the following specific attributes:

■ Server name

■ Port

■ Profile name

■ Profile DBnr

■ Profile Fnr

For Base APDs the following specific associations exists:

■ **"Has component FI "with association code CF:**
   which Data Definition Modelues belongs to this Application description.

■ **"Has component PR "with association code CP:**
   which Natural Programming Objects belongs to this Application description.

■ **"Has library SY "with association code LI:**
   each library of Natural Programming Objects, that are linked to the Base APD with association "Has component PR", must be linked to the Base APD.

> **Note:** The association "Has library SY" is built automatically when changing the association "Has component PR".

Documenting Compound Applications Descriptions

Compound Application Descriptions are documented as objects of type System with system type O. Compound Application Descriptions have no specific attributes.

The association from Compound APD to Base APD is named "Has subappl. SY" with association code CS.

> **Note:** Predict Maintenance functions ensure that only Systems of system type B are linked to Compound APDs.

Documenting Data Definition Modules (DDM)

Data Definition Modules are documented as objects of type File. The association from a Base APD to the Data Definition Modules is named "Has component FI" with association code CF.

Documenting Natural Programming Objects

Natural Programming Objects are documented as objects of type Program with corresponding program type.

The association from a Base APD to the Natural Programming Objects is named "Has component PR" with association code CP.

The following rules apply for Natural Programming Objects linked to a Base APD:

- The implementation pointer must be full qualified.
- All members must be on the same Natural System File.
- For each Natural Programming Object the library this object is in must be documented as a System of system type A with the association "Has library SY" (association code LI).
- If database number or file number of the implementation pointer is changed, the Natural Programming Object is removed from the Base APD. If it is also the last member with the library in this Base APD, the corresponding system type A, representing the library, must be removed from association "Has library SY" too.

Documenting Libraries

Libraries are documented as objects of type System with system type A. The association from a Base APD to the libraries is named "Has library SY" with association code LI.