Kofax Customer Communications Manager

5.0

ITP/SDK AS/400

Manual ITP/SDK AS/400

KOFAX
from Lexmark

Contents

Introduction

With the ITP/SDK for iSeries a DID Developer can create and maintain a DID on the iSeries. This manual introduces you to this ITP/SDK for iSeries. The DID is stored in a database on the iSeries. The ITP/SDK for iSeries also aids in the generation of iSeries data retrieval programs. When a DID is finished it can be downloaded to a PC where the ITP/MDK Repository cliënt 3.5.18 or up is installed. Otherwise you need ITP/Workstation to download the DID to the PC. If you are not familiar with the basic ITP and DID concepts, we advise you to read chapter DID concepts (page 7) and chapter ITP concepts for an introduction.

This ITP/SDK for iSeries manual describes the tasks involved in creating and maintaining a DID on the iSeries that result in adding and changing data in the ITP/SDK for iSeries database and the creation of actual iSeries objects. Besides this, it is also a reference manual.

All documentation can also be found on the Kofax Customer Communications Manager Knowledge Center (http://ccmkc.kofax.com).

About the SDK 400 Manual

This ITP/SDK for iSeries manual describes the tasks involved in creating and maintaining a DID on the iSeries that result in adding and changing data in the ITP/SDK for iSeries database and/or the creation of actual iSeries objects. Besides this, it is also a reference manual.

A short word on the installation and configuration of ITP/SDK for iSeries can be found in chapter Installation and configuration (page 17).

In the chapter Tasks (page 20) more information can be found about the structure of a DID and how to create a DID with ITP/SDK iSeries.

Chapter Reference (page 27) provides information on how to work with the ITP/SDK iSeries.

Copyrights and trademarks

transmission is restricted by any applicable U.S. regulation or statute, without the prior written consent, if required, of the Bureau of Export Administration of the U.S. Department of Commerce, or such other governmental entity as may have jurisdiction over such export or transmission. You represent and warrant that you are not located in, under the control of, or a national or resident of any such country.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

DID concepts

This is the concept reference. Purpose of this manual is to provide you a complete description of all the ITP concepts and their interaction and dependencies involved in DID development (ITP terminology). This is an introduction into these concepts. This manual does not contain a step-by-step explanation or formal definition.

Since this manual is targeted at DID developers the description of the concepts refers to things that they need and are familiar with. It is sometimes important however to recognize that the users of your DID, the Model developers, have a different view and other or less information than you have. Therefore, the description of a concept for Model developers is also included in this manual, wherever that is relevant.

DID

A DID (**D**atabase **I**nterface **D**efinition) is a description of (a part of) a database. ITP uses the abstract concept of Entries (page 8) for files or tables. Relations between database entities can be expressed through Entry-Subentry (page 12) relations. Apart from Entries, DIDs can also contain Functions (page 11). A DID itself does nothing. It is used to create Models. During Model creation, the information that is needed is copied to the Model. When a Model is run ITP retrieves the data from the database, based on the information that is stored in the Model.

A DID is created from a DID Document (ITP/SDK MultiPlatform) or DID Administration (ITP/SDK for iSeries). Within the document or administration we distinguish two levels: the DID level and the DID Module (ITP/SDK MultiPlatform) or DID Source (page 8) (ITP/SDK for iSeries) level. The DID level describes the interface to the DID. On the DID Module/Source level you actually define the Entries, Subentries and functions.

On the DID level you can designate some Entries as Main Entries (page 14). Only the Main Entries in a DID can be accessed directly. All other Entries have to be accessed through a (series of) Entry-Subentry relation(s). Functions that are defined in DID Sources have to be declared on the DID level before they can be used in a Model.

A DID can contain multiple DID Sources. In the DID Source you define the components i.e., Entries and Functions. On the DID level you declare which of these components are accessible in ITP Models. The DID Sources can contain the complete description of a subset of your database, which can be reused in several DIDs. The DID Sources can be combined differently in different DIDs. This enables you to give different sets of users or Model Developers a different view of your database. These different views can be used to provide a more logical view or to control authorizations.

Model developers view

A DID (Database **I**nterface **D**efinition) is a description of (a part of) a database. ITP uses the abstract concept of Entries (page 8) for files or tables. Relations between database entities are expressed through Entry-Subentry (page 12). Apart from Entries, DIDs can also contain Functions (page 11). A DID itself does nothing. It is used in the Data Retrieval part of the Data Backbone to fill the field sets with data that is needed in the templates and the content. When a Model is run ITP retrieves the data from the database, based on the information that is stored in the Data

Backbone.

DID module

DID Modules are used in the ITP DID Language in the ITP/SDK MultiPlatform. Within a DID (page 7) the Entries (page 8) and Functions (page 11) are defined in the DID Module. On the DID level you only specify which Main Entries (page 14) and Functions are available to a Model Developer. A DID has one or more DID Modules. The DID Modules only have a meaning for the DID Developer. The Model Developer does not know anything about DID Modules and does not see a difference between Entries that are defined in different DID Modules.

For each DID Module you have to specify a Connection Type. You can use DID Modules for different Connection Types in a single DID. You can link Entries together from different DID Modules through the Import (page 9)-Export (page 9) mechanism. You can create DID Modules for different ITP Connection types and combine these DID Modules in a DID. This way you can create true MultiPlatform DIDs. This feature is only supported by the ITP/SDK MultiPlatform.

DID Source

DID Sources are used in the ITP/SDK for iSeries. Within a DID (page 7) the Entries (page 8) and Functions (page 11) are defined in the DID Source. On the DID level you only specify which Main Entries (page 14) and Functions are available to a Model Developer. A DID has one or more DID Sources. The DID Sources only have a meaning for the DID Developer. The Model Developer does not know anything about DID Sources and does not see a difference between Entries that are defined in different DID Sources.

You can link Entries together from different DID Sources through the Import (page 9)-Export (page 9) mechanism. With the ITP/SDK for iSeries you can only create DIDs for the ITP iSeries Connection. DID Sources do not have a connection attribute.

Entry

An Entry enables a Model Developer to retrieve a set of data in an ITP Model. The Entry describes what data are retrieved and how they are retrieved. An Entry definition consists minimally of a set of formal parameters (page 9) (possibly empty), a set of Fields (page 9) and a Data Retrieval (page 14) method. If an Entry is used as a Main Entry (page 14) that has formal parameters, the definition should also include a Key Retrieval (page 15) method. From the point of view of a DID Developer an Entry is an output mechanism; it describes the output of the Data Retrieval method.

The sets of Fields that are retrieved by the Entry are called records.

The formal parameters describe the input parameters of the Data Retrieval. The Fields describe the output parameters. The Key Retrieval method is used when the user should be able to select a record. The Key Retrieval method results in a set of Fields that is used as the actual parameters (page 10) in the Data Retrieval method.

The set of data that an Entry retrieves can conform to (a part of) a database record, e.g., a table in an SQL database or a physical or logical file on the iSeries. An Entry does not have to coincide with an actual table in the database, but can present a pre-processed, simplified or summarized view of a single table or a collection of tables.

Model developers view

An Entry enables you to retrieve a set of data in a Data Backbone. The Entry describes which formal parameters (page 9) are needed for the retrieval and it describes the Fields (page 9) that are retrieved. An Entry can be compared with a table or file in a database. The formal parameters are the keys and the Entry Fields are the Fields in the file or table. An Entry is an input mechanism for a Data Backbone.

Export

Entries (page 8) defined in a DID-source (page 8) can be used as a Subentry (page 12) in other DID-sources. Entries that are to be used in another DID-source should be explicitly exported, and before they can be used in another DID Module they have to be imported (page 9) in that DID-source.

Import

Entries defined in another DID-source (page 8) can be used as a Subentry (page 12). These entries from another DID-source must be explicitly imported . Before an Entry can be imported it has to be exported (page 9) from the DID-source in which it is defined

Field

A Field is an element of the data set that is retrieved by an Entry (page 8). The data set that is retrieved can only be accessed through these Fields. Each Field has a data type. The possible data types for Fields are a subset of the data types allowed in the databases associated with a Connection Type. In ITP Language, numerical Fields are always mapped to the NUMBER type of ITP Language and alphanumerical Fields are always mapped to the TEXT type of ITP Language. The values are converted by ITP during evaluation of the Model. Information about the original data type is lost in this conversion.

Model Developer View

A Field is an element of the data set that is retrieved by an Entry (page 8). The data set that is retrieved can only be accessed through these Fields. Each Field has a data type. The possible data types for Fields are NUMBER and TEXT.

Not all numeric Fields and text Fields are the same in a database. Each text Field in a database has a maximum length and for each numeric Field in a database the maximum length and the number of decimal positions is administrated. This information is stored in the DID and can be viewed. ITP converts these Fields however to its own NUMBER and TEXT formats. ITP Language does not have means to retrieve information about the original format and does not treat the numeric Field formats differently.

Formal parameter

The term parameter is in general used for both the definition of the argument that should be passed to an Entry, program, procedure, function or subroutine and for the argument that is passed. This poses a problem.

If I define a mathematical function f(x) = x + 2, the identifier 'x' is said to be 'the parameter' of the function f, meaning that it indicates the argument that should be passed. If I write down an application of the function f, e.g. f(3) 3 is also said to be 'the parameter' of the function f, meaning that it is the argument that is actually passed to the function f. When I talk about 'the parameter' of the function f it is not clear whether I mean x or 3.

In order to distinguish between these two uses of the word parameter we call an identifier that is used in the function definition to denote an argument that should be passed the formal parameter (x in f(x) = x + 2). An expression that yields the argument that is passed is called an actual parameter (3 in f(3) or a+b in f(a+b)).

A Formal Parameter is a Field that is needed to determine exactly which data set should be retrieved from the database. The definition of the formal parameters is stored in the Entry definition or function definition in the DID. Formal parameters have the same attributes as a field (page 9) and in most instances, each formal parameter is also included as a Field of that Entry. It is then recommended that the formal parameters are defined exactly the same as the Field, including the name. The order in which the formal parameters are defined is significant.

> **Note**
> The maximum number of formal parameters per entry is 23.

Model Developer View

A Formal parameter is a Field that is needed to determine exactly which data set should be retrieved from the database. The definition of the formal parameters is stored in the Entry definition or function definition in the DID. Formal parameters have the same attributes as a field (page 9) and in most instances, each formal parameter is also included as a Field of that Entry. In most cases, you do not have to worry about the formal parameters during the development of the ITP Models. If the Entry is a Main Entry, the actual parameters will be provided through Key Selection (page 15). If the Entry is a Subentry, the actual parameters are defined in the Entry-Subentry definition. Actual parameters can also be provided or overridden in an Entry application.

Actual parameter

The term parameter is in general used for both the definition of the argument that *should be* passed to an Entry, program, procedure, function or subroutine and for the argument that *is* passed. This poses a problem.

If I define a mathematical function f(x) = x + 2, the identifier 'x' is said to be 'the parameter' of the function f, meaning that it indicates the argument that should be passed. If I write down an application of the function f, e.g. f(3) 3 is also said to be 'the parameter' of the function f, meaning that it is the argument that is actually passed to the function f. When I talk about 'the parameter' of the function f it is not clear whether I mean x or 3.

In order to distinguish between these two uses of the word parameter we call an identifier that is used in the function definition to denote an argument that should be passed the Formal parameter (x in f(x) = x + 2). An expression that yields the argument that is passed is called an Actual parameter (3 in f(3) or a+b in f(a+b)).

An Actual parameter for an Entry application can be provided in several manners.

If the Entry is a Main Entry, the actual parameters will be provided through Key Selection (page 15). The Key Retrieval (page 15) method defined with the Entry is called. This method yields a set of actual parameters that will be used for the data retrieval (page 14) method.

If the Entry is a Subentry, the actual parameters are defined in the Entry-Subentry (page 12) definition.

Actual parameters can always be provided or overridden by the PAR keyword in a WITH or FORALL instruction in ITP Language.

Model Developer View

An Actual parameter for an Entry application can be provided in several manners.

If the Entry is a Main Entry, the actual parameters will be provided through Key Selection (page 15).

If the Entry is a Subentry, the actual parameters are defined in the Entry-Subentry (page 12) definition.

Actual parameters can always be provided or overridden by the keyword PAR in an instruction WITH or FORALL in ITP Language.

DID Defined function

A DID Defined Function is a function that is defined in the DID. The function performs an operation on a host system. Each DID Defined function has zero or more formal parameters (page 9) and a single result. The method that is to be called when the function is accessed is also defined in the DID.

The operations that a DID Defined Function can perform depend on the Connection Type. You could for example use a DID Defined Function in order to calculate totals, subtotals or averages or use a DID Defined Function to update records in the database.

A function can also technically be seen as an Entry that yields exactly one record that has a single Field.

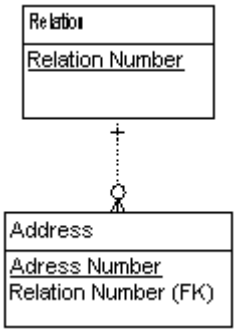
A DID Defined function is also called a Standard User Defined Function, User Defined Function or Standard Function.

Model Developer View

A DID Defined Function is a function that is defined in the DID. The function performs some operation on a host system. Each DID Defined function has zero or more formal parameters (page 9) and a single result. In a Data Backbone you should treat these functions similar to the ITP standard functions.

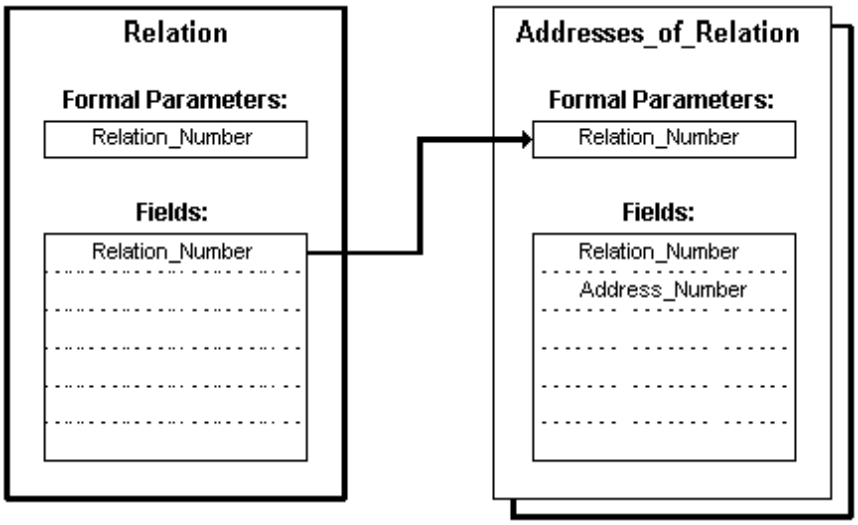
Subentry

Example



The above picture shows part of an example database specification. The example contains two entities: an entity Relation and an entity Address. Each Address belongs to a specific Relation. A Relation can have more than one Address.

There is a one-to-many relation between the entities Relation and Address. To express this relation in the DID you can define two Entries; an Entry Relation and an Entry Addresses_of_Relation. If you want to use the Entry Addresses_of_Relation, you need to specify the relation number for which you want to retrieve the addresses. This value is available in the Field Relation_Number of the Entry Relation. In the subentry definition, you can specify that the Field Relation_Number should be used as the actual parameter of the Entry Addresses_of_Relation.



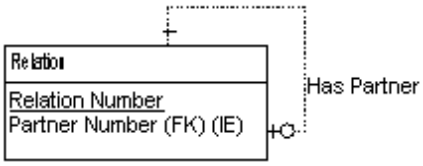
In a Subentry definition, you express the relation between the Entry and the Subentry, by supplying actual parameters for the formal parameters of the Subentry. You can supply Fields or constants for the actual parameters.

Model Developer View

A Subentry expresses a relation between two Entries. A DID for a relation database might contain an Entry for relations and an Entry for addresses. With the help of a Subentry, you can automatically obtain the addresses belonging to a relation.

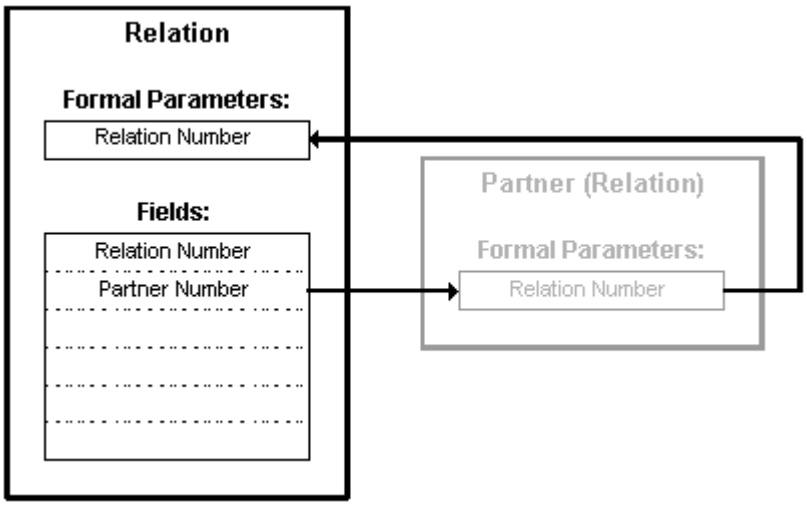
Alias

Example

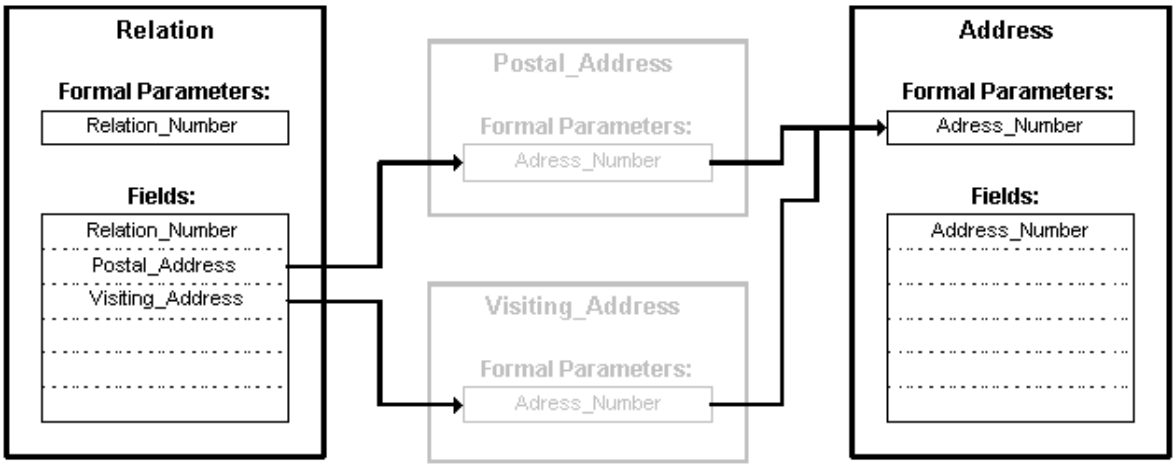


The above picture shows part of an example database specification. The example contains an entity Relation. There is a recursive relation contained in this entity. Each Relation can have an optional Partner that also is a Relation.

The entity Relation contains a recursive relation "is partner of". To express this relation in the DID you can define an Alias for an Entry. Such an Alias is an alternative name for an Entry. The Alias Entry can be used as a Subentry. We can now write `WITH Partner IN Relation.Partner` instead of `WITH Partner IN Relation.Relation`. The first alternative is clearer.



An Alias can also be useful if you want to use a certain Entry twice as a Subentry. For example, an Entry Relation has a Field Postal_Address and a Field Visiting_Address. Both Fields contain an address number. This address number serves as a formal parameter for an Entry Address. You could then make two Aliases Postal_Address and Visiting_Address for the Entry Address. You can then use these Aliases as Subentries in the Entry Relation. For the Subentry Postal_Address, you define the Field Postal_Address as the actual parameter. For the Subentry Visiting_Address, you define the Field Visiting_Address as the actual parameter. This is shown in the picture below.



An Alias is an alternative name for an Entry.

Main entry

A Main Entry is an Entry that can be used as a starting point for the data retrieval in a Data Backbone. Most Data Backbones use only one or two Main Entries. All other Entries are accessed as Subentries. Data Backbone Developers can only use Main Entries or Entries that are accessible as Subentries. All other Entries that are defined in a DID Module cannot be used by the Developer. Which Entries can be used as Main Entry is defined in the DID.

Model Developer View

A Main Entry is an Entry that can be used as a starting point for the data retrieval. Most Data Backbones use only one or two Main Entries. All other Entries are accessed as Subentries.

Data retrieval

An entry describes the input parameters and the output parameters for the Data Retrieval. Data Retrieval is the core of the ITP Data-Text merge process. The formal parameters of an Entry describe the input parameters of the Data Retrieval method. The Fields of an Entry describe the output parameters of the Data Retrieval method. Which method should be called is defined in the Entry. ITP has no knowledge what the Data Retrieval method does or how it performs its operations. ITP will accept any Data Retrieval method that obeys the calling convention, parameter definitions and field definitions.

Once the actual parameters for an Entry (page 8) are known the Data Retrieval method is started, obeying the calling conventions of the Connection Type. The actual parameters of the entry are passed to the Data Retrieval method. The actual parameters can be provided through several

different mechanisms:

1. The actual parameters are supplied as the result of Key Selection (page 15); Main Entries only.
2. The actual parameters are supplied through the Subentry (page 12) mechanism; Subentries only.
3. The actual parameters are supplied through the mechanism PATH/PAR of the ITP Language.

Depending on the connection type one or more mechanisms are available for data retrieval. Some mechanisms make use of ITP internals, other mechanisms use programs or SQL queries. The result of the data retrieval method should always be a data set (record) that obeys the Field definitions of the Entry.

The Connection Types are implemented in the ITP DataManager. Every time that Data Retrieval is required in a Model, the Data Retrieval method and parameters are sent to the ITP DataManager. The ITP Data Manager takes care of all communication involved in executing the Data Retrieval method and receiving the results. The ITP environment only communicates to external databases through the DataManager.

Key selection

When a Model is run ITP/OnLine or ITP/Workstation will show a Key Selection screen if:

- the Model encounters a Main Entry (page 14) that has one or more formal parameters (page 9) and
- the Entry is accessed with the keyword IN, and
- the parameters were not supplied through the Run Model API.

In this screen the users sees a list of records, these records are retrieved by the Key Retrieval (page 15) method. Each of these records shows a number of Fields that should enable the user to select a unique record. After the users selects a record ITP obtains the actual parameters (page 10) from the Key Retrieval method. These actual parameters are used to call the Data Retrieval (page 14) method. Which Fields are shown in the selection screen and which Fields are used as actual parameters is determined by the Entry definition.

Model Developer View

When a Model is run ITP/OnLine or ITP/Workstation will show a *Key Selection* screen if:

- the Model encounters a Main Entry (page 14) that has one or more formal parameters (page 9) and
- the Entry is accessed with the IN keyword and
- the parameters were not supplied through the Run Model API.

In this screen the users sees a list of records. Each of these records shows a number of Fields that should enable the user to select a unique record. After the user selects a record, ITP will use the Fields of this record as actual parameters (page 10) to retrieve a data set.

Key retrieval

The method Key Retrieval controls the Key Selection (page 15). It determines which Fields are shown in a Key Selection screen and which actual parameters are passed to the Data Retrieval (page 14) method. Key Retrieval is a variant of Data Retrieval.

Depending on the connection type one or more mechanisms are available for key and data retrieval. Some mechanisms make use of ITP internals, other mechanisms use programs or SQL queries. The result of the Key Retrieval method should always be a data set (record) that obeys the Parameter and Field definitions of the Entry.

The Connection Types are implemented in the ITP DataManager. Every time that Key Retrieval is required in a Model, the Key Retrieval method and parameters are sent to the ITP DataManager. The ITP Data Manager takes care of all communication involved in executing the Key Retrieval method and receiving the results. The ITP environment only communicates to external databases through the DataManager.

Installation and configuration

The tasks involved in creating and maintaining a DID on the iSeries result in adding and changing data in the ITP/SDK for iSeries database and/or the creation of actual iSeries objects. Any actual iSeries objects involved are referred to in the ITP/SDK for iSeries database, but it is important to realize that any changes made to these objects outside the context of the ITP/SDK for iSeries may go unnoticed by the ITP/SDK for iSeries. The most important iSeries objects involved are the source libraries and object libraries associated with a DID-source, the data retrieval programs associated with an Entry and the programs execute function associated with User-defined functions.

Installation, configuration, upgrading and licensing

Installation

The ITP/SDK for iSeries can be installed along with the ITP iSeries Connection software. Refer to the appropriate ITP Installation manuals for the details on the installation.

Configuration

No additional configuration is needed for the ITP/SDK for iSeries.

Upgrading

DIDs created in the ITP/SDK for iSeries version 2.1 can be copied into the ITP/SDK for iSeries version 3.5 database through the command COPYDID (page 74).

The following example copies the DID XYZ and all associated DID Sources from database version 2.1 to database version 3.5:

```
ITPPGM35/COPYDID
   FROMDID(XYZ)
   TODIDLIB(ITPSRC35)
   FROMSRCLIB(ITPSRC21)
   TOSRCLIB(ITPSRC35)
   REPLACEDB(*ADD)
   REPLACEDID(*REPLACE)
```

Licensing

The ITP/SDK for iSeries is installed as an iSeries licensed program. Refer to the appropriate ITP Installation manuals for the details on the installation. All ITP/SDK for iSeries objects are shipped as part of the ITP/SDK for iSeries licensed program, with the user profile ITPOWNER as object owner and the public authority set to *CHANGE. Refer to the appropriate ITP Installation manuals for the details on the ITPOWNER user profile.

Work with DIDs

When speaking about a DID in the context of the ITP/SDK for iSeries one of two concepts is meant:

1. The DID as the whole of the Database Interface Definition; that which can be downloaded to the PC to create and run ITP Models with.
2. The DID as the set of DID level data in the ITP/SDK for iSeries database, often referred to as the DID information.

The DID as a whole consists of:

- the DID information
- the data of all the DID-Sources that are linked to the DID

The DID-information consists of:

- the primary DID attributes e.g., the name of the DID, an identification
- the set of DID-Sources linked to the DID
- the set of Main entries of the DID
- the set of User-defined functions of the DID

From the panel "Work with DIDs" the DID information can be manipulated.

Function Keys

F6=Create

Create a new DID-record and set its primary attributes.

Options

2=Change
Change the primary attributes of the DID.

4=Delete
Delete the DID-record. The sets of Main Entries and User-defined functions with the DID are deleted as well. A DID cannot be deleted as long as there are DID-sources linked to it; these links must be explicitly deleted (refer to "DID/DID-source links (page 38)").

5=Display
Display the primary attributes of the DID.

11=Main entries
Displays the panel Work with Main entries (page 38) from which the Main entries of the DID can be manipulated.

15=Functions
Displays the **(DID level)** panel Work with User defined functions (page 39) from which the User-defined functions of the DID can be manipulated.

16=DID-sources

Displays the panel Work with DID-sources of DID (page 38) from which the DID-sources linked to the DID can be manipulated.

24=Save DID components
Displays the menu Save DID components (page 39) from which the DID can be saved to tape and/or save file.

26=Check DID
Performs a consistency and correctness check on the DID. If the DID is consistent and correct its status is set to "F" (Final), otherwise its status is set to "I" (Inconsistent) and messages will be issued to explain the nature of the errors found. The messages will be in the English language only.

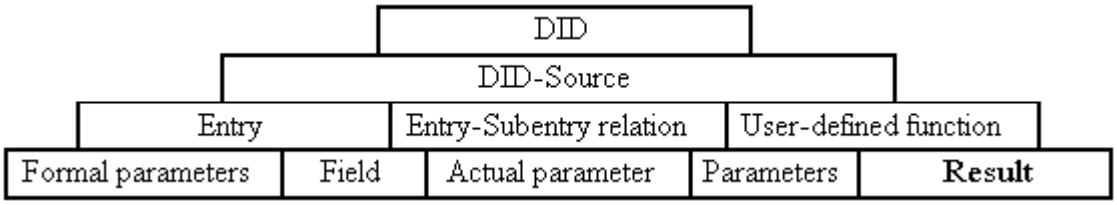
The primary attributes of the DID information are:

DID name	A case sensitive name of maximal 10 characters that serves as a unique identification of the DID inside the ITP/SDK for iSeries.
Identification	A three letter code used as identification of the DID in ITP Model documents.
Description	A descriptive text
Status	Can be either Inconsistent (I) or Final (F). The status can be changed as a result of checking the DID with option "26=Check DID" or manually changed using option "2=Change".
Creation date	Automatically set when the DID-information record is created with function key F6=Create.
Date of last check	Automatically set when the DID is checked with option "26=Check DID"
Time of last check	Automatically set when the DID is checked with option "26=Check DID"

Tasks

Overview

A DID is organized in several layers of information. The ITP/SDK for iSeries follows this organization closely.



From every layer down a 1:n relation is possible: a DID can have 1 or more DID-sources linked to it, A DID-source may contain one or more Entries or User defined functions. A DID and a DID-source are the only items that may exist in the ITP/SDK for iSeries without relation to another SDK-item, but they will then only serve as intermediate results of the DID development process. Entries, Entry-Subentry relations and User-defined functions can only exist inside a DID-source, Formal parameters and Fields can only exist within an Entry.

Options

2=Change

Change the attributes of the Result of the User-defined function.

5=Display

Display the attributes of the Result.

The attributes of the Result are:

Result name — A case sensitive name of maximal 30 characters. This name must start with an uppercase and may not contain any blanks.

Data type — The data type of the Result. This can be TEXT (T), PACKED (P), ZONED (Z), BINARY (B) or Unicode (W). ITP does not support all DDS field types: refer to "Supported DDS field types (page 72)" for more information.

Length — The length in bytes of the Result.

Decimal positions — Number of decimal positions in the Result. Only used if the data type of the Result is numeric (PACKED, ZONED or BINARY)

Alignment — Alignment of the Result. This can be either left aligned (L) or right aligned (R). Only used if the data type of the Result is not numeric

(TEXT or Unicode)

Getting started

The different layers in a DID and thus the ITP/SDK for iSeries are best introduced by the fastest way to create a working DID:

Step	Action	Result	Level
1	Start the ITP/SDK for iSeries (page 27) by entering the command `ITPPGM35/STRITP` on a command line.	The ITP Main menu (page 30) is displayed.	
2	Choose option 30. ITP Quick start (page 31).	The Create DID display is prompted.	DID
3	Enter a name for the DID, a 3-letter identification and a descriptive text.	The DID is created. The Create DID-source display is prompted.	DID
4	Enter a name for the DID Source, the names for two iSeries libraries (they may be the same library), and a descriptive text	The DID Source and the libraries are created. The Link DID Source display is prompted.	DID Source level
5	The name of the DID Source from the previous step is already filled in, press Enter.	The DID Source is linked to the DID. The ITP Main menu (page 30) is displayed.	DID

Two major items of the ITP administration are now set up: a DID and a DID Source. All the work that remains will be done in the context of either the DID or the DID Source.

In the context of the DID Source:

Step	Action	Result	Level
6	From the ITP Main menu (page 30) choose option "2. Work with DID Sources (page 34)".	The panel Work with DID Sources (page 34) is displayed.	DID Source level
7	Choose option "16=Work with DID-source" for the subfile record containing the DID Source created in step 4.	The DID-source menu (page 41) is displayed.	
8	Choose option "1. Work with Entries (page 42)"	The panel Work with Entries (page 42) is displayed.	Entry level
9	Press the function key F20 = Generate.	The display Generate Entry (page 46) is prompted.	
10	Enter a name for the Entry (starting with	The Entry is generated.	

Step	Action	Result	Level
	an uppercase letter and containing no blanks) and the library and file you want to describe. Do not change the defaults already filled in for the other fields.	The Work with Entry fields panel is displayed.	
11	Type option "7=Convert to valid ITP name" at the first field displayed on the subfile and press the key F13 = Repeat. Then press Enter	All field names that ITP extracted from the file are converted to valid ITP names. The Work with Entry fields panel is still displayed.	Field level
12	Press the function key F20 = Add fields to entry.	The Fields are added to the Entry. If the file is not keyed the Create Main entry panel is prompted (the next step is skipped). If a file is keyed the panel Work with Formal parameters (page 64) is displayed.	
13	Press the function key F20 = Add formal parameters to entry.	The Formal parameters are added to the Entry. The panel Work with Entries (page 42) is displayed.	
14	Press the function key F3 = Exit	The menu DID-source is displayed.	Entry level
15	Press the function key F3 = Exit	The panel Work with DID-sources (page 34) is displayed.	DID Source level
16	Press the function key F3 = Exit	The ITP Main menu (page 30) is displayed.	

After step 13 the Entry is created and can be used. More Entries, User defined functions and Entry-Subentry relations may be added to the DID-source.

Before an Entry can be used in an ITP Model it must be "accessible in the DID". There are two ways in which an Entry is accessible in the DID:

1. The Entry is Subentry to an Entry that is already accessible in the DID;
2. The Entry is a Main entry of the DID.

The registration of an Entry as a main entry is done in the DID context:

Step	Action	Result	Level
17	From the ITP Main menu (page 30) choose option "1. Work with DIDs (page 18)".	The. Work with DIDs (page 18) panel is displayed.	DID-level
18	Choose option "11=Main entries" for the subfile record containing the DID created in step 3.	The Work with Main entries (page 38) panel is displayed.	

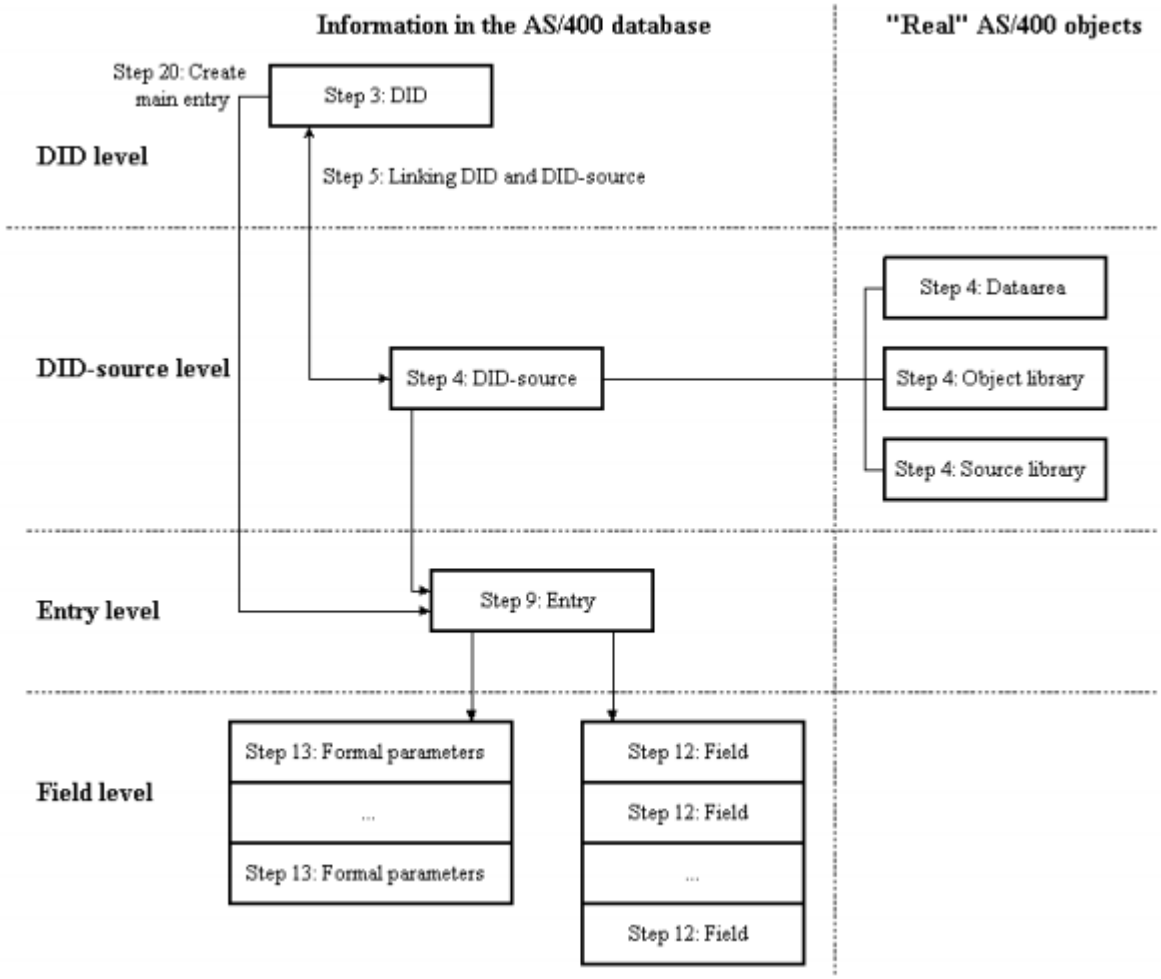
Step	Action	Result	Level
19	Press function key "F6=Create".	The Create Main entry display is prompted.	
20	Enter the name of the Entry (created in step 10) and the name of the DID-source (created in step 4) and press Enter.	The newly created Entry is registered as a Main entry of the DID. The Work with Main entries (page 38) panel is displayed.	
21	Press function key "F3=Exit"	The Work with DIDs (page 18) panel is displayed.	
22	Press function key "F3=Exit"	The ITP Main menu (page 30) is displayed.	

After step 20 the DID is finished and ready for download. The download must either be initiated from a PC on which the ITP/MDK Repository cliënt version 3.5.18 or up is installed or from a PC on which ITP/Workstation is installed.

For download using the ITP/MDK Repository, refer to the online help that comes with the action "Download ISeries DID...".

For download using ITP/Workstation, please refer to the iSeries DID Management chapter from the ITP Administrator manual for more information about how to proceed from this point on.

The previous steps resulted in:



ITP/SDK for iSeries architecture

The architecture of the DIDs and the ITP/SDK for iSeries is closely related to the way databases and applications on the iSeries are mostly organized:

ITP concepts		**iSeries concepts**			
DID		Database		Application	
DID Source		Library		Library	
Entry		File and Record format		Program	
Field	Formal parameter	Field	Key field	Result parameter	Input parameter

Concepts

Like a database and an application on the iSeries ITP's DID is essentially an abstract entity. Inside the ITP/SDK for iSeries the DID only exists as a coherent set of descriptions stored in several tables of the ITP/SDK for iSeries database. The DID actually becomes an object when it is downloaded to a PC for further use in creating and running ITP Models.

More than one iSeries library may be involved in one application and one library can even be part of more than one application, just like more than one DID-source may be linked to one DID and a DID-source may be linked to one ore more DIDs. A DID-source itself mostly (but not necessary) describes the data in one iSeries library.

A DID-source is less abstract than a DID in the aspect that a DID-source refers to actual objects in the form of one or more iSeries libraries. These libraries contain the sources and programs that may be needed to do the actual work involved in using Entries and User-defined functions when running an ITP Model.

Similar to (physical and logical) files and programs on the iSeries that are always located inside a library, Entries and User-defined functions can only exist inside a DID-source.

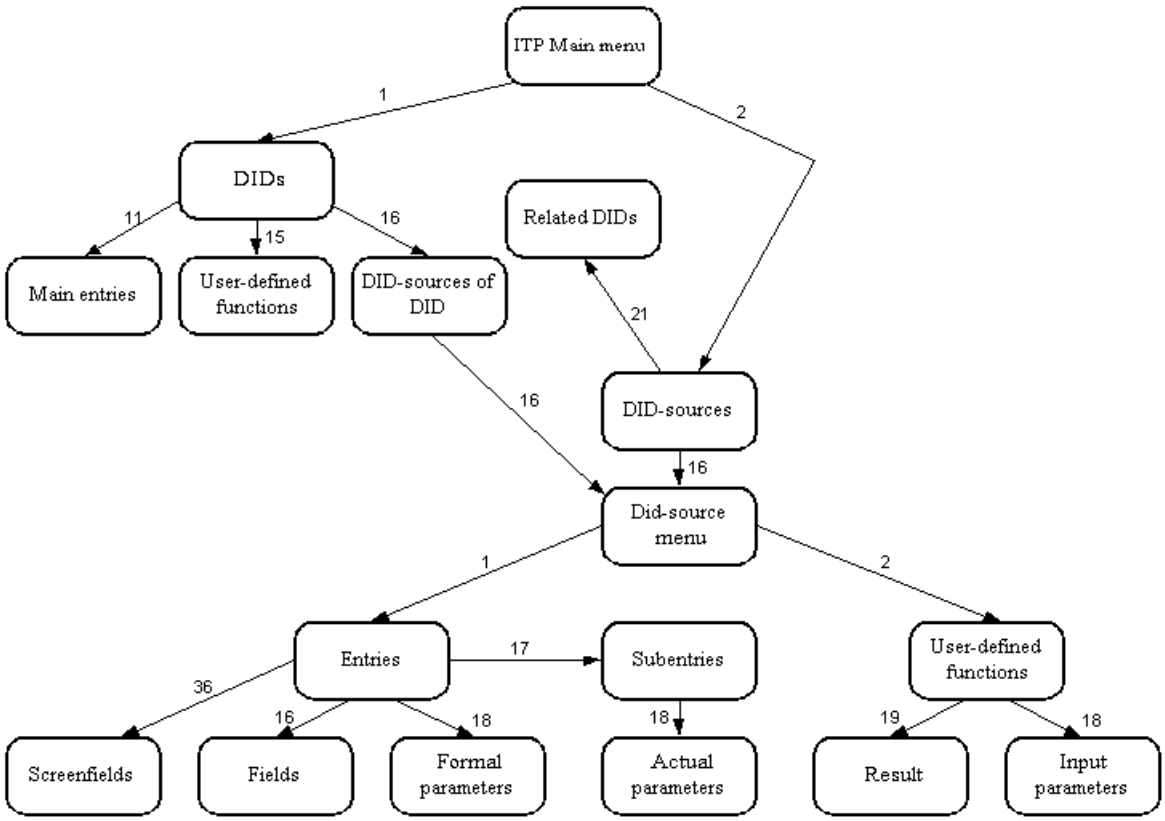
An Entry is mostly ITP's description of a logical or physical file: the Entry fields describe the fields in the file and the Formal parameters describe the key fields of the file. Unlike logical or physical files that are basically of a static nature an Entry also has a dynamic aspect. As an Entry is used for the specific purpose of retrieving data from a database when used in an ITP Model, it can also be considered to be the description of the interface of a program which performs the function "retrieve all information described by the Entry fields and identified by the Formal parameters from the database and return the results to the ITP Model".

A User-defined function is in many aspects similar to an Entry. The only differences are that a User-defined function has exactly one Result field and the User-defined function is intended to describe a program that performs some action other than the retrieval of data.

Entry fields and Formal parameters are ITP's equivalent to the fields and key fields of a (physical or logical) file or the result and input parameters of a program.

Programs

As can be concluded from the previous paragraph, the ITP concepts like the iSeries concepts are hierarchically organised. This hierarchy may be observed in the program structure of the ITP/SDK for iSeries as well:



The boxes represent the most important "Work with" panels and the ITP Main menu (page 30) and DID-source menu (page 41). The arrows indicate that one display can be reached from another. The numbers with the arrows indicate the option numbers that must be used.

Reference

Start the ITP/SDK for iSeries

You can start the ITP/SDK for iSeries with the command:

```
ITPPGM35/STRITP
```

The command will set the library list needed for the ITP/SDK for iSeries and displays the ITP Main menu (page 30).

The parameters of the command are:

LANGUAGE The language used on the display.

`Values:`

***DFT** This value is the command default. Refer to ITP Management (page 33) about setting the default language. After installation of the ITP/SDK for iSeries the default language is English

The default language is used.

DEU German is used as the language on the display.

ENG English is used as the language on the display.

ESP Spanish is used as the language on the display. This language is not yet implemented.

FRA French is used as the language on the display. This language is not yet implemented.

ITA Italian is used as the language on the display. This language is not yet implemented.

NLD Dutch is used as the language on the display.

MENUOPTION The ITP Main menu option that is called.

`Values:`

0 This value is the command default.

No option is called. The ITP Main menu is displayed.

1 The option "1. Work with DIDs" is called.

2	The option "2. Work with DID-sources" is called.
7	The option "7. Authorization menu options ITP Main menu" is called.
30	The option "30. ITP Quick start" is called.
40	The option "40. ITP Management" is called.
90	The option "90. Signoff" is called.

SRCLIB The library containing the ITP/SDK for iSeries database

`Values:`	***DFT**	This value is the command default. The default library is used. For the ITP/SDK for iSeries 3.5 this is the library ITPSRC35.
	Library name	Any valid iSeries library name. The library must exist and contain the ITP/SDK for iSeries database.

All parameters only affect the behavior of the ITP/SDK for iSeries session started with the command. Other users and any ITP/SDK for iSeries session (already started or not) are not affected by the choice of parameters.

Library list

When starting the ITP/SDK for iSeries the current library list used is saved. When leaving the ITP/SDK for iSeries the library list is restored to the value prior to starting the ITP/SDK for iSeries. All library list changes while working with the ITP/SDK for iSeries (by the SDK or the user itself) are rolled back.

The ITP/SDK for iSeries itself is responsible for implementing any library list changes needed. When starting the ITP/SDK for iSeries the ITP program library and the ITP/SDK for iSeries database library used are added to the end of the library list. If QTEMP was not already in the library list it is added to the end as well. When generating data retrieval programs for Entries (at the moment the compiler is invoked to create the program) the library containing the database file over which the Entry has been generated and the library in which the program has to be created (the object library of the DID-source) are temporarily added to the library list as well.

The command `STRITP` also uses the library ITPPGM35 as product library.

Typical library list settings are:

Before starting SDK	After starting SDK	During program generation	After program generation	After ending SDK
<System library list>	<System library list>	<System library list>	<System library list>	<System library list>
	ITPPGM35	ITPPGM35	ITPPGM35	
<User library list>	<User library list> ITPPGM35 ITPSRC35	<Database library> <Object library> <User library list> ITPPGM35 ITPSRC35	<User library list> ITPPGM35 ITPSRC35	<User library list>

All these library list changes are automatically performed by the ITP/SDK for iSeries. This also means that at least four library list entries (five if QTEMP is not part of the user library list) must be available in order to work with the ITP/SDK for iSeries. The Database library will not be added to the library list if the Database library is already part of the User library list.

Beware that if an action is performed in the ITP/SDK for iSeries that is not under direct control of the SDK, the library list might not be the one that is actually needed. If for instance a data retrieval program is compiled manually (after using "F10=Command Entry" from one of the ITP/SDK for iSeries screens) the library containing the database file is not added to the library list. Also the presence of the "F10=Command Entry" on many of the ITP/SDK for iSeries screens enables the setting of the library list in such a way that the SDK may no longer function properly (there is no "restore library list" after the "F10=Command Entry" has ended).

Suppose the library XLIB is manually added to the library list before the program generation, then the library list settings from the previous example will be:

Before starting SDK	After starting SDK	After manually changing the library list	During program generation	After program generation	After ending SDK
<System lib. list>	<System lib. list>	<System lib. list>	<System lib. list>	<System lib. list>	<System lib. list>
	ITPPGM35	ITPPGM35	ITPPGM35	ITPPGM35	

Before starting SDK	After starting SDK	After manually changing the library list	During program generation	After program generation	After ending SDK
<User library list>	<User library list> ITPPGM35 ITPSRC35	XLIB <User library list> ITPPGM35 ITPSRC35	<Database lib.> <Object library> XLIB <User library list> ITPPGM35 ITPSRC35	XLIB <User library list> ITPPGM35 ITPSRC35	<User library list>

ITP Main menu

```
When the ITP/SDK for iSeries is started the ITP Main menu is displayed:

MNUITP                        ITP Main menu                     1/09/09
14:32:59

 Select one of the following:

       1. Work with DIDs (page 18)
       2. Work with DID-sources (page 34)




       7. Authorization menu options ITP Main menu (page 31)


      30. ITP Quick start (page 31)
      40. ITP Management (page 33)

      90. Signoff

 Option:
 ===>

 F3=Exit             F12=Cancel          F10=Command entry
```

DIDs and DID-sources

With the ITP/SDK for iSeries a DID Developer can create and maintain both DIDs and DID-sources on the iSeries. DIDs are the functional units with which ITP models can be created and run (after downloading it to the PC). DID-sources are the technical descriptions of (a part of) an application in terms of Entries (and their relations) and User-defined functions. The DID-sources may be linked to DIDs. The DID sources are only significant for the DID Developer.

The Model Developer does not know anything about DID sources and does not see a difference between Entries that are defined in different DID Sources.

Refer to Work with DIDs (page 18) and Work with DID-sources (page 34) for more information.

Authorization

Apart from the native iSeries authorization of objects the ITP/SDK for iSeries has some extra security features. The first one is the possibility to change the authorization of the iSeries commands that are called when an option on the ITP Main menu is chosen by using option "7. Authorization menu options ITP Main menu" on the ITP Main menu.

ITP Quick start

ITP Quick start is a fast and easy way to set-up the basic parts needed in a DID. It is a wizard-alike sequence of ITP/SDK for iSeries functions that create a DID, a DID-source, and link the DID-source to the DID. Refer to ITP Quick start (page 31) for more information.

ITP Management

ITP Management offers an interface on the ITP Libraries, the setting of user specific defaults, an interface to display the PTF level of the ITP/SDK for iSeries licensed program, and an option to set the default language of the SDK.

Refer to ITP Management (page 33) for more information.

Authorization menu options ITP Main menu

The options on the menu Authorize menu options that is displayed after selecting option "7. Authorization menu options ITP Main menu" on the ITP Main menu (page 30) prompt the iSeries command `EDTOBJAUT`. With these options the object authority for certain user(-groups) on the different commands of the ITP main menu can be changed.

```
MNUITPS                 Authorize menu options

 Select one of the following:

       1. Authorize       Work with DIDs
       2. Authorize       Work with DID-sources




       7. Authorize       Authorisation menu options ITP Main menu


      30. Authorize       ITP Quick start
      40. Authorize       ITP Management

      90. Signoff

 Option:
 ===>

 F3=Exit             F12=Cancel          F10=Command entry
```

By default the authority of the commands from the ITP Main menu (page 30) are set to be owned by the user profile ITPOWNER (that has *ALL authority for the command) and *PUBLIC authority is set to *CHANGE.

ITP Quick start

ITP Quick start is a fast and easy way to set-up the basic parts needed in a DID. It is a wizard-alike sequence of ITP/SDK for iSeries functions that create a DID, a DID-source, and link the DID-source to the DID. The ITP Quick starts a sequence of three SDK functions:

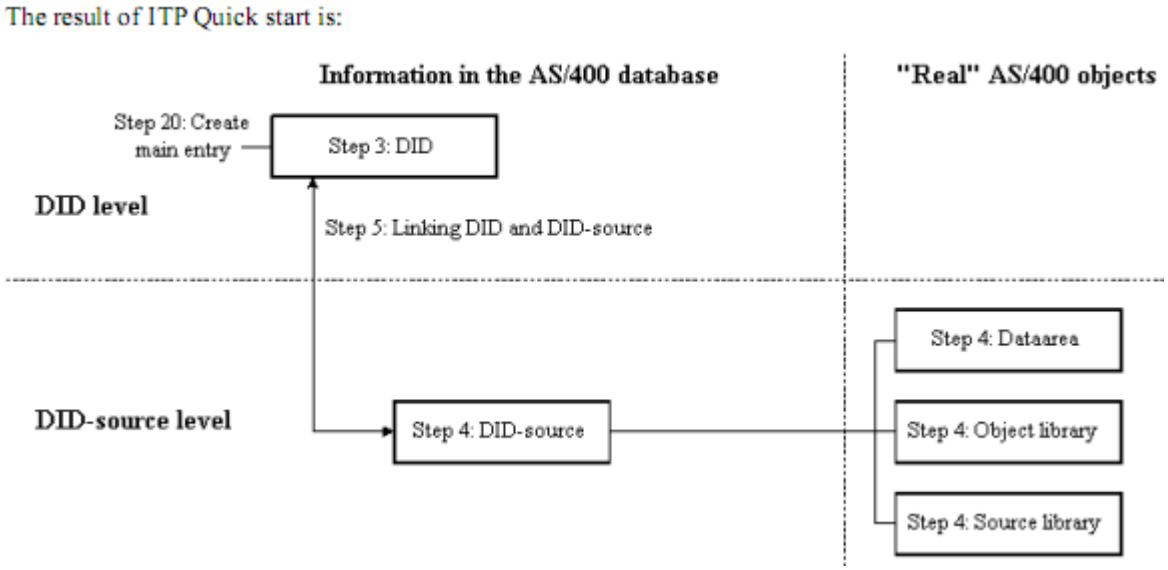
1. Create DID.
2. Create DID-Source.
3. Link DID-Source to DID.

At any given time when the display with one of these functions is prompted the function keys "F3=Exit" and "F12=Cancel" can be pressed. This will result in the cancellation of the ITP Quick start. Beware that there is no rollback performed, all items already created remain in existence. If for instance F3 is pressed when the Create DID-source display is prompted, the DID information is already created in step 1, and will still exist after ITP Quick start is cancelled.

More information about creating a DID can be found in Work with DIDs (page 18).

More information on creating a DID-source can be found in Work with DID-sources (page 34).

More information on linking DID-sources to DIDs can be found in DID/DID-source links (page 38).



Refer to the listing in the paragraph Getting Started (page 21) for a detailed description of every step.

ITP Management

Using option "40. ITP Management" on the ITP Main menu (page 30) displays the menu ITP Management.

```
MNUPKMAN                    ITP Management

 Select one of the following:

       1. Work with ITP libraries
       2. Work with User options
       3. Display PTF level


       6. ITP license information
       7. Set default language




      90. Signoff

 Option:
 ===>

 F3=Exit             F12=Cancel          F10=Command entry
```

From this menu the configuration settings for the ITP/SDK for iSeries can be set or viewed. Note that some of the programs behind the options require special authorization.

ITP Libraries

The ITP/SDKfor iSeries consists of several iSeries libraries. The option "1. Work with ITP libraries" displays a list of these libraries and offers some useful iSeries functions to manipulate the libraries. Beware that the libraries shown in the list are the libraries actually used by the SDK at the moment the list is displayed. Refer to Start the ITP/SDK for iSeries (page 27) for information on how to use another ITP/SDK for iSeries database library.

User options

The ITP/SDK for iSeries offers some user specific preferences, which can be stored for determining default values. Currently, only defaults for the generation of data retrieval programs can be stored; option "25=Generate Data retrieval" on the panel Work with Entries (page 42). Defaults are stored per user profile. If the current user has no defaults specified, the settings from the special *DEFAULT are used. Defaults can be created, changed, deleted, except for the special *DEFAULT, and displayed on the panel "Work with User defined options GENPGMDTA" that is displayed after option "2. Work with User options" is used on the menu "ITP Management".

PTF level

The ITP/SDK for iSeries is distributed as an iSeries licensed program. PTFs for the ITP/SDK for iSeries are distributed in the same way PTFs for other iSerieslicensed products are distributed. Option "3. Display PTF level" displays the current status of the PTFs for the SDK.

Default language

The ITP/SDK for iSeries is a multilingual product. Several different languages are supported. With the option "7. Set default language" the default language for the SDK can be set. Currently, only German (DEU), English (ENG), and Dutch (NLD) are supported. Note that this setting only takes effect after the SDK is restarted. Refer to Start the ITP/SDK for iSeries (page 27) for information on how to use a language other than the default when starting the SDK.

Work with DID Sources

In the context of the ITPSDK for iSeries a DID Source is ITP's interface on (a part of) the data that can be accessed by an ITP Model that uses the DID to which the DID Source is linked. A DID Source is intended to describe a coherent part of the entire application that is described in a DID. A DID Source on the iSeries serves the same purpose as a DID Module in the ITP/SDK MultiPlatform.

The DID Source consists of:

- the primary DID source information e.g., the name of the DID-source, the names of associated iSeries libraries
- an iSeries data area with the same name as the DID-source used to set authorities on the DID Source and some default setting
- one or two iSeries libraries used to store the programs and sources referred to in the DID Source
- the contents of the DID Source:
  - the set of Entries in the DID Source
  - the set of User-defined functions in the DID Source

- the relations between the Entries in the DID Source
- the imported entries from other DID Sources

From the Work with DID Sources panel the DID Source information can be manipulated.

Function keys

F6=Create

Creates a new DID-source record and set its primary attributes. Also creates the iSeries data area and the iSeries libraries associated with the DID-source if they do not already exist. The function asks for confirmation if any of the referred libraries is already associated with another DID-source

Options

2=Change

Change the primary attributes of the DID-source

3=Copy

Copies a DID-source including the iSeries data area and its contents (the Entries, User-defined functions etc.). The iSeries libraries associated with the DID-source are only copied if the new iSeries libraries did not already exist. The function asks for confirmation if any of the referred libraries for the new DID-source is already associated with another DID-source.

4=Delete

Delete the DID-source including the iSeries data area and all of its contents. On a confirmation panel the user is prompted to choose whether or not the iSeries libraries associated with the DID-source must be deleted as well.

Beware that no warning is given if the iSeries libraries are also associated with another DID-source.

A DID-source cannot be deleted as long as it is linked to a DID; the link must be explicitly deleted (refer to "DID/DID-source links (page 38)").

5=Display

Display the primary attributes of the DID-source

9=Save

Enables you to save the DID-source (to save file or other media). The complete DID-source is saved: the primary information, the data area, the contents of the DID-source, and the associated libraries.

10=Restore

Enables you to restore the DID-source (from save file or other media). The complete DID-source is restored: the primary information, the data area, the contents of the DID-source, and the associated libraries.

Beware that any existing DID-source information and objects are overwritten by what is restored.

Option "16= Work with DID-source" Displays the "DID-source menu (page 41)". This is the starting point to work with the contents of the DID-source.

14=Authority

Executes the EDTOBJAUT command on the data area with the same name as the DID-source. The data area is assumed to reside in the library containing the SDK database. Refer to "DID-source data area (page 37)" for more information.

21=Work with related DIDs

Displays the "Work with related DIDs of DID-source (page 38)" panel. This panel shows all the DIDs the DID-source is linked to. Only the primary DID information may be viewed from this panel, and the link between DID-source and DID may be deleted.

26=Check DID-source

Performs a consistency and correctness check on the DID-source. If the DID-source is consistent and correct its status is set to "C" (Consistent), otherwise its status is set to "I" (Inconsistent) and messages will be issued to explain the nature of the errors found. The messages will be in the English language only.

The primary attributes of the DID-source information are:

DID-source name — A name of maximal ten characters that serves as a unique identification of the DID-source inside the ITP/SDK for iSeries. The name must be a valid iSeries object name.

Object library — The name of an iSeries library in which the data retrieval programs for Entries and the execute function programs for User-defined functions are stored.

Note that this library must exist for the DID-source to be consistent, but the library may be empty when all Entries in the DID-source use *DBMS400 and no User-defined functions are present in the DID-source.

Source library — The name of an iSeries library in which the sources for the program objects in the Object library are stored. The source library may be the same library as the Object library.

Description — A descriptive text

Status — Can be either Inconsistent (I) or Consistent (C). The status can be changed as a result of checking the DID-source with option "26=Check DID-source" or manually changed using option "2=Change".

Creation date — Automatically set when the DID-source information record is created with function key F6=Create.

Date of last check	Automatically set when the DID is checked with option "26=Check DID-source"
Time of last check	Automatically set when the DID is checked with option "26=Check DID-source"
Date of last save	Automatically set when the DID is saved with option "9=Save"
Time of last restore	Automatically set when the DID is restored with option "10=Restore"

DID-source data area

When a DID-source is created an iSeries dataarea with the same name as the DID-source is also created. This data area is created in the current database library for the ITP/SDK for iSeries (ITPSRC35 by default) and is used for two purposes:

1. Authorization of the attributes and contents of a DID-source;
2. To store some System values specific to the DID-source.

Refer to "DID-source menu (page 41)" for more information on the System values specific to a DID-source.

Most of the ITP/SDK for iSeries programs that access a DID-source check for the existence of the associated data area. If the data area exists, the programs check the authority the user has for this data area to find out if the user is allowed to perform the action on the DID-source. The mechanism does not limit the access on the iSeries libraries associated with the DID-sources, nor the authority on the DIDs linked to the DID-source. The Check DID-source program also does not check with the data area for authority. The following table shows the behavior of the SDK with a particular authority on the data area.

**EXCLUDE*	An extra message "NOT AUTHORIZED" is displayed on the Work with DID-sources panel. The user is not allowed to perform any action that changes the contents of the DID-source.
**USE*	The user may browse the attributes and the contents of the DID-source, but is not allowed to change them.
**CHANGE*	The user may change the attributes of the DID-source and the attributes of the contents of the DID-source (the Entries and User-defined functions) and is allowed full access on the Entry-Subentry relations in the DID-source.
**ALL*	The user has full access to the existing attributes and contents of the DID-source. This is equal to the access that is granted to the user when the data area does not exist.

If the data area does not exist it can be created manually with the command:

```
CRTDTAARA DTAARA(ITPSRC35/[DID-source]) TYPE(*CHAR) LEN(20)
```

DID/DID Source links

DIDs are the functional units with which ITP Models can be created and run, after downloading the DID to the PC. DID Sources are the technical descriptions of (a part of) an application in terms of Entries (and their relations) and User-defined functions. The ITP/SDK for iSeries contains two ways of accessing the links between DIDs and DID Sources:

1. Panel "Work with DID Sources of DID",
   from the panel Work with DIDs (page 18) choose option "16=DID-sources".
2. Panel "Work with related DIDs of DID-source",
   from the panel Work with DID-sources (page 34) choose option "21=Work with related DIDs".

Function keys

F6=Link

Only on the "Work with DID Sources of DID" panel. Creates the link between a DID and a DID Source.

Options

4=Delete

On both the "Work with DID-sources of DID" and the panel "Work with related DIDs of DID-source". Deletes the link between the DID and the DID Source. Note that any Main entries and User-defined functions from the linked DID Source remain registered with the DID. These registrations must explicitly be deleted from the DID./ For more information, refer to Work with Main entries (page 38) and Work with User defined functions (page 39).

5=Display

On the panel "Work with DID-sources of DID" displays the primary attributes of the DID Source.

On the panel "Work with related DIDs of DID-source" displays the primary attributes of the DID.

16=Work with DID Source

Displays the DID-source menu (page 41). This is the starting point to work with the contents of the DID Source.

Refer to DID-sources and modularity (page 56) for more information on the ideas behind, and the details of the use of multiple DID Sources in one DID.

Work with Main entries

Entries are defined in a DID Source. An Entry can only be used in an ITP Model if:

1. it is a Main entry of a DID the ITP Model is based upon, or
2. it is used as a Subentry of the Entry it is a Subentry of.

To be able to traverse through the database in an ITP Model (via the Entry-Subentry relations in the DID) a starting point is needed. In ITP terms these starting points are called Main entries. Only Entries from DID Sources linked to the DID may be registered as Main entries for this DID. Beware

that if a link between a DID and DID Source is deleted the Main entries are not affected and should be deleted explicitly.

Function keys

F6=Create

Registers an Entry from one of the linked DID-sources as a Main entry for the DID.

Options

4=Delete

Deletes the registration of an Entry as Main entry.

5=Display

Displays the primary attributes of the Entry.

Work with User defined functions (DID level)

User-defined functions are defined in a DID Source. User-defined functions can only be used in an ITP Model if they are explicitly registered with the DID the ITP Model uses. Only User-defined functions from DID Sources linked to the DID may be registered with this DID. Beware that if a link between a DID and DID Source is deleted the User-defined functions are not affected and should be deleted explicitly.

Function keys

F6=Create

Registers a User-defined function from one of the linked DID-sources in the DID.

Options

4=Delete

Deletes the registration of a User-defined function from the DID.

5=Display

Displays the primary attributes of the User-defined function.

Save DID components

From the Save DID components menu the DID and it contents (including all linked DID-sources) can be saved.

Options

1. `Save DID`

Saves the DID and its contents, including all DID-sources linked to the DID, but excluding the

Object and Source libraries of the linked DID-sources.

4. Save object libraries
Saves all the Object libraries of all the DID-sources linked to the DID.

`Save DID` operates in three stages:

1. The Save DID panel is prompted.
    a. Enter the name of an iSeries save file.
    This is only a temporary save file that will be created automatically and placed in QTEMP. Its only purpose is to bundle the DID information stored in multiple files in the ITP/SDK for iSeries database into one iSeries object.
    b. Enter the 'device' where the save file from 1.a. must be stored on.
2. The iSeries command `SAVOBJ` is prompted.
    a. Enter the save-options you want to change, like the iSeries target release.
    This save command applies to the DID information from the SDK database that is stored into the temporary save file from 1.a.
3. The iSeries command `SAVOBJ` is prompted.
    This save command applies to the saving of the temporary save file named in 1.a. to the 'device' chosen in 1.b.
    a. Enter the name of an iSeries save file if the DID must be stored in a save file.
    If a save file is chosen in 1.b., then the name of an already existing save file (not in QTEMP and thus different from the one named in 1.a.) must be named here.
    b. Enter the save-options you want to change, like the iSeries target release.

To restore a DID previously stored by `Save DID` the command `RSTDID` is available. This command can be called from any command line inside the SDK, and operates in three stages similar to the `Save DID`:

1. The panel Restore DID is prompted.
    a. Enter the name of the DID to restore.
    b. Enter the name of an iSeries save file.
    This is only a temporary save file that will be created automatically and placed in QTEMP. Its only purpose is to bundle the DID information stored in multiple files in the ITP/SDK for iSeries database into one iSeries object.
    c. Enter the 'device' the save file must be restored from.
2. The iSeries command `RSTOBJ` is prompted. This is the equivalent of the third stage of `Save DID` and restores the temporary save file.
    a. Enter the restore-options you want to change, like the name of the save file the temporary save file was stored in.
3. The iSeries command `RSTOBJ` is prompted.
    This restore command applies to the restoring of the DID information from the temporary save file named in 1.b.
    a. Enter the restore-options you want to change, like the library to restore the DID information to.
    b. After the DID information is restored to the library chosen in step 3.a, the DID-s information is copied into the SDK database to complete the restore action.

Save Object Libraries prompts the iSeries command `SAVLIB`. The Object libraries of all the DID-sources linked to the DID are already filled in. All the parameters of the command `SAVLIB` may be changed. Note that a library occurs multiple times in the list of libraries to save if this library is used as Object library by more than one of the DID-sources linked to the DID.

DID-source menu

The "DID-source menu" is the starting point to work with the contents of the DID Source.

Options

1. Work with Entries:
Displays the panel Work with Entries (page 42) from which the Entries and their relations in the DID Source can be created, changed, deleted, etc.

2. Work with User defined functions:
Displays the panel Work with User defined functions (page 39) from which the User-defined functions in the DID Source can be created, changed, deleted, etc.

5. Work with System values:
Prompts the display "Work with Default values" on which some DID Source specific system values may be entered.

8. Work with Object library:
Starts WRKLIBPDM on the Object library of the DID Source.

9. Work with Source library:
Starts WRKLIBPDM on the Source library of the DID Source.

Work with System values

When working with the Entries in a DID Source, the ITP/SDK for iSeries itself can (re-) generate Entries based upon an existing (physical or logical) database file. The System values of the DID Source influence this (re-) generation.

Default file library There is no default value for this setting.

This is the library where the database files to describe in the DID-source are assumed to be located. The library name entered here is used as the default library name in the programs that (re-) generate entries. On these (re-) generation programs it is always possible to change the library actually used.

ITP fieldname default The default value for this setting is *TEXT.

This setting determines whether the DDS names (value *NAME) or the DDS texts (value *TEXT) are used as the basis for the ITP field names of the Entry.

The system values are stored in the DID Source data area. If no DID Source data area exists, no DID Source specific system-values can be entered and the default values are used. Refer to "DID-source data area" to see how the data area can be created manually. This topic also explains how the data area can be used to set authorization on the DID Source.

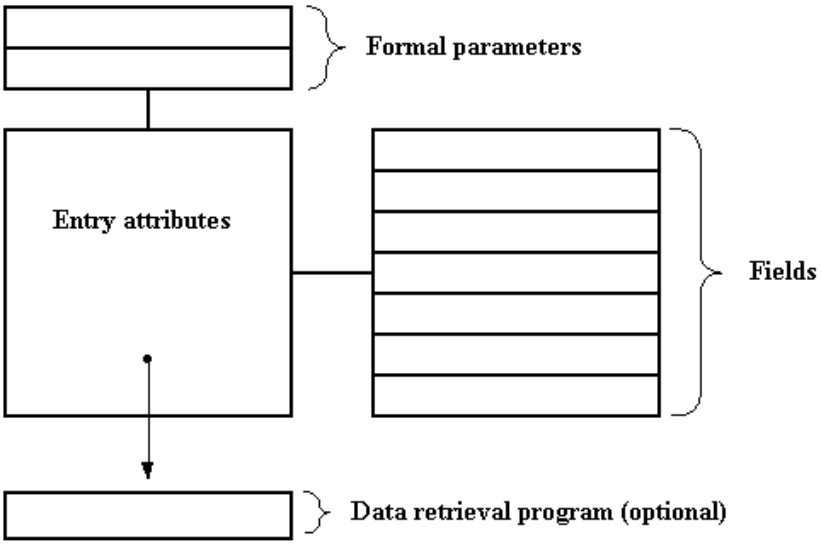
Work with Entries

Entries are the central concept in ITP DID construction. From the point of view of ITP Model construction an Entry describes the data that can be retrieved from the database with one single data retrieval action. In this aspect the Entry parallels the description of a database file or table. Most Entries will in fact be no more than the description of one file or table from the database, but the Entry concept is much broader that that.

An Entry consists of:

- the primary Entry attributes , e.g. the name of the Entry, or its type
- the extended Entry attributes (only if the Entry was generated)
- the set of Formal parameters
- the set of Fields
- (optionally) the set of Screen fields
- (optionally) a reference to an iSeries program

When comparing an Entry to a database file or table the Formal parameters parallel the key fields of a file or table, while the Entry Fields parallel the fields in that file or table. When considering the more dynamic aspects of an Entry it can best be looked at as the description of a program: "given the values of the Formal parameters all records in the database that are identified by these values must be retrieved and the values described by the Fields must be passed to ITP".



From the panel "Work with Entries" Entries can be created, changed, deleted etcetera, and Entry-Subentries relations can be manipulated:

Function keys

F6=Create

Create a new Entry record and set its primary attributes. It is probably easier to generate an Entry using function key "F20=Generate". (refer to "Entry Creation and Generation (page 46)" for more information), as generation also includes the steps of creating Fields and Formal parameters.

F20=Generate

Generate a new Entry using an existing database file. This function not only creates the Entry

record, but also starts the program to define the Fields of the Entry and the program to define the Formal parameters of the Entry if the file is keyed. Refer to "Entry Creation and Generation (page 46)" for more information.

F22=Import

Import an Entry from another DID-source. Refer to "DID-sources and modularity (page 56)" for more information on the ideas behind and the details of the use of multiple DID-sources in one DID.

Options

2=Change

Change the primary attributes of the Entry. When function key "F14=More information" is pressed on the "Change Entry" panel the extended attributes of the Entry can be changed.

3=Copy

Copy an Entry including its Formal parameters and Fields. The copy action may be to another DID-source. Any referenced data retrieval program is not copied.

4=Delete

Delete an Entry (including its Formal parameters, Fields, Subentry relations, etc.). Any referenced data retrieval program is not deleted. A window with a warning and an extra request for confirmation is displayed if the Entry is still used as a Subentry or an Alias is based on it.

5=Display

Display the primary attributes of the Entry. When function key "F14=More information" is pressed on the "Display Entry" panel the extended attributes of the Entry can be displayed.

14=Aliases

Displays the panel Work with Aliases of entry (page 58) that lists all Entries that are defined as Alias of the Entry.

15=Export

Mark an Entry as available for Import in another DID-source. This changes the Export attribute of the Entry. The Export attribute of the Entry may be changed by using option "2=Change" as well. Refer to DID-sources and modularity (page 56) for more information on the ideas behind and the details of the use of multiple DID-sources in one DID.

16=Fields

Displays the panel Work with Entry fields (page 60) that lists all Fields of the Entry.

17=Sub entries

Displays the panel Work with Sub entries (page 58) that lists all Subentries of the Entry.

18=Formals

Displays the panel Work with Formal parameters (page 64) that lists all Formal parameters of the Entry.

19=Calc. Offsets

Based on the sequence numbers and field lengths the offsets of the Fields of the Entry are recalculated. The record length attribute of the Entry is set accordingly. Refer to Work with Entry fields (page 60) for more information on the Fields and their attributes.

25=Generate Data retrieval

Generates the source for the data retrieval program of the Entry and compiles this to a program. Only allowed for generated Entries. Refer to Entry Creation and Generation (page 46) for more information.

27=Work with Data retrieval

Starts WRKOBJPDM for the data retrieval program object in the Object library of the DID-source.

30=Linked DIDs

Displays a window listing all DIDs where the Entry is used as a Main entry.

31=Where as Sub entry

Displays a window listing all Entries where the Entry is used as a Subentry.

36=Screen fields

Displays the panel Seq. numbers screen fields PGM key retrieval (page 54) that lists all Fields of the Entry. On this panel sequence numbers can be used with the Fields to indicate that they must be displayed in the key-selection window on the PC. Refer to Entry Creation and Generation (page 46) for more information

61=Check re-generation of entry

Checks what will happen if the Entry would be regenerated and lists the changes in a spool file. Using function key "F4=Prompt" with the option prompts a panel on which another database file may be selected. After the results are calculated and written to a spool file this spool file is displayed. Leaving the display spooled file prompts a panel where can be indicated whether the spool file must be deleted, printed or both. Refer to Entry Creation and Generation (page 46) for more information.

62=Re-generate entry

Regenerates the Entry while maintaining any Aliases and Subentry relations. Using function key F4=Prompt with the option prompts a panel on which another database file may be selected. After the Entry is regenerated the offsets of the Fields of the Entry are recalculated. Note that any manual changes to the original Entry, like previously dropped Formal parameters, need to be reapplied after regeneration. Refer to Entry Creation and Generation (page 46) for more information.

There are several possibilities to control the Entries that are visible on the Work with Entries panel. Entering a value in the list selection fields or pressing one of the list selection function keys has precedence over any options entered on the subfile (they will be ignored). Both the list selection fields as the function keys act as restrictors on the items displayed: only those items that satisfy the conditions are part of the list displayed. Using one of the function keys changes the value of the "Entry kind" selection field and displays a message indicating the type of list displayed. Changing the value of the list selection fields after using one of the function keys only further restricts the Entries displayed within the restriction imposed on the list by the function key. A restriction activated by one of the function keys can only be changed by pressing one of the other list selection function keys.

List selection `Name of the Entry`	Only the Entries with a name that is alphabetically greater than or equal to the name entered here are part of the list.
List selection `Entry kind`	A blank indicates that the list is not restricted to Entries of one kind. A **D** indicates that only Definitions Entries are part of the list, an **A** restricts the list to the Alias Entries only, and an **I** restricts the list to only the Entries imported from another DID-source.
Function key `F17=Exports`	Only the Entries that are labelled for possible export to another DID-source are part of the list.
Function key `F18=Non exports`	Only the Entries that are not labeled for possible export to another DID-source are part of the list.
Function key `F19=All entries`	Both the Entries that are and that are not labeled for possible export to another DID-source are part of the list.

The primary attributes of an Entry are:

Entry name	A case sensitive name of maximal 30 characters that serves as a unique identification of the Entry inside the DID-source. This name must start with an uppercase and may not contain any blanks.
Entry type	Can be either "S" (Singular) or "P" (Plural). Singular means that the Entry is meant to retrieve at most one record from the database and must be used in a WITH construct in the Model construction language. Plural means that the Entry is meant to retrieve none, one or more records from the database and must be used in a FORALL construct in the Model construction language. Beware that this attribute has no influence on the workings of the actual data retrieval. Refer to Entry Creation and Generation (page 46) for more information.
Record length	This indicates the total number of bytes per database record that is returned to an ITP Model that uses the Entry. This equals the record length of the file unless the field definitions, field offsets or field lengths of the Entry were manually changed.

PGM data retrieval The special value `*DBMS400` or the name of an iSeries program. `*DBMS400` indicates that ITP itself does the actual data retrieval when an ITP Model that uses this Entry is run. Otherwise the program indicated is called to do the data retrieval and send the data back to ITP. Refer to Entry Creation and Generation (page 46) for more information.

Export Can be either "Y" (Export) or "N" (Non export). This tag indicates whether the Entry may be imported in another DID-source or not.

Entry generation Can be either "A" (Automatically), "M" (Manually), or blank (Automatically but changed). This tag indicates the way in which the Entry was created.

Description A descriptive text

The extended attributes of an Entry are only used if the Entry was generated.

Generation date The date on which the Entry was generated

Generation time The time on which the Entry was generated

File The name of the file the Entry was generated over

Library The library where the file was in when the Entry was generated

Record format The name of the record format of the file the Entry was generated over

Entry Creation and Generation

There are two ways to create Entries in the ITP/SDK for iSeries:

a. Use the key F6 = Create on the panel Work with Entries (page 42) to create Entries manually. Every Entry attribute, all Field and Formal parameter definitions, and the program that does the actual data retrieval must be specified and build completely manually. The SDK does not offer any automatic support during Entry creation.
b. Use the key F20 = Generate on the panel Work with Entries (page 42) to create Entries based on the definition of a logical or physical file. This function reads the definition of the file and will automatically prompt the displays from where Field and Formal parameter definitions are entered.

To create or generate an Entry starting from the panel Work with Entries (page 42) the following steps are needed:

Step	Action	Create	Generate	Level
1	Define the Entry	Press the function key F6 = Create.	Press the function key F20 = Generate.	Entry-level
		The panel Create Entry is prompted.	The panel Generate Entry is prompted.	

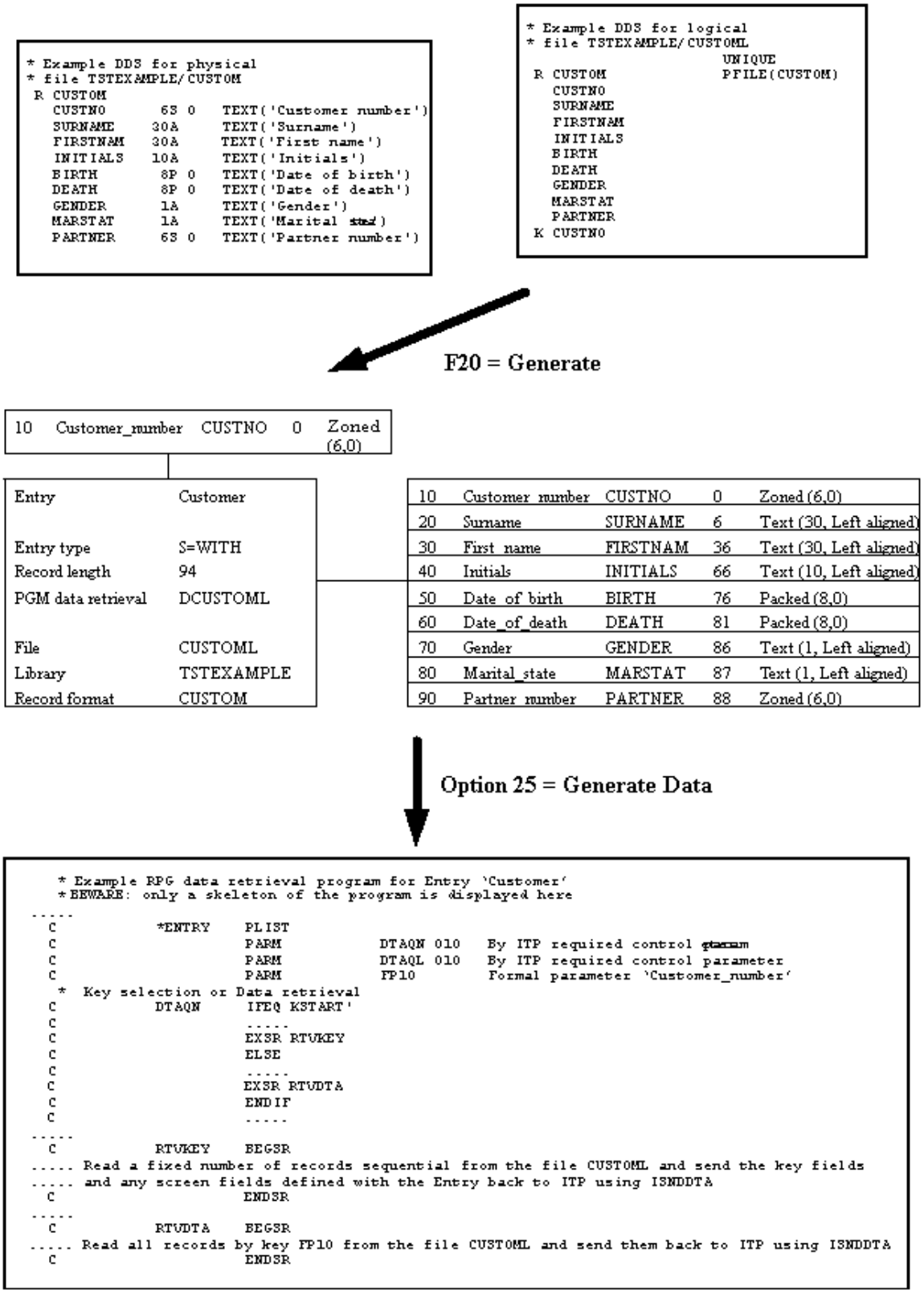
Step	Action	Create	Generate	Level
		Set the Entry attributes and press Enter.	Set the Entry attributes and press Enter.	
2	Define Entry fields	The panel Work with Entries (page 42) is displayed.		Field-level
		Use option "16=Fields".		
		A panel "Work with Entry fields" is displayed.	A panel "Work with Entry fields" is prompted.	
		Create the Field definitions using the function key F6 = Create.	Change the name of the Fields using option "7=Convert to valid ITP name", possibly in conjunction with the function key F13 = Repeat.	
		Press the function key F3 = Exit.	Press the function key F20 = Add fields to entry.	
3	Define Formal parameters	The panel Work with Entries (page 42) is displayed.		Field-level
		Use option "18=Formals".		
		A panel "Work with Formal parameters" is prompted.	A panel "Work with Formal parameters" is prompted.	
		Create the Formal parameters definitions using the function key F6 = Create.	Change the Formal parameter definitions of the Entry using the options "4=Drop key field" and "6=Retrieve key field".	
		Press the function key F3 = Exit.	Press the function key F20 = Add formal parameters to entry.	
4	Set the Screen fields (optionally)	The panel Work with Entries (page 42) is displayed.		Field-level
		Use option "36=Screen fields".		
		The panel Seq. Numbers screen fields PGM key retrieval (page 54) is prompted.		
		Enter sequence numbers for any additional field to be displayed in the Key selection window (page 54).		
		Press the function key F3 = Exit.		

Step	Action	Create	Generate	Level
5	Build the data retrieval program	Build a data retrieval program according to the specifications in Entry Creation and Generation/Data retrieval (page 51).	If *DBMS400 was chosen as data retrieval interface you don't need to build a retrieval program. Or Generate a program using option "25=Generate Data retrieval". Or Build a data retrieval program according to the specifications in Entry Creation and Generation/Data retrieval (page 51).	Entry-level

Note
Pressing the function keys F3 = Exit or F12 = Cancel during Entry generation cancels the generation process but all items already created remain in existence.

Using the function key F20 = Generate is the most common way to create an Entry. The generation of an Entry is a wizard-like sequence of SDK functions that enables you to either choose *DBMS400 to do the actual data retrieval for the Entry, or let the SDK generate a data retrieval program. Even when manual changes to the Entry are needed, starting with a generation of the Entry and applying the manual changes afterwards is often the best way to proceed. Manual creation of an Entry requires in-depth knowledge of the way ITP handles Entries.

If an Entry is generated and a data retrieval program is registered with the Entry, this program can be generated in a separate step by using option "25=Generate Data retrieval" on the display Work with Entries (page 42). The following diagram best illustrates the entire generation process:



The Entry that is generated consists of Formal parameters (optional), Entry attributes and Fields.

From the point of view of the Entry generation process an Entry is just a different way of

representing a file definition. In this aspect an Entry is no more than storing the file definitions in a way that makes it better accessible to ITP with one extra feature: instead of the DDS names to identify the file and its fields the DDS text can be used as the basis for identification in ITP. Refer to DID-source menu/Work with System values (page 41) for more information on DDS names versus DDS text. Refer to Work with Entries (page 42) for more information on the attributes of the Entry itself. Refer to Work with Entry fields (page 60) and Work with Formal parameters (page 64) for more information on Entry Fields and Formal parameters.

Work with Entry fields and Work with Formal parameters

The panels "Work with Entry fields" and "Work with Formal parameters" mentioned in this section are special versions of the ones mentioned in the Work with entries section. Their functionality is restricted. Refer to "Work with Entry fields (page 60)" and "Work with Formal parameters (page 64)" for more general information on the subjects of Fields and Formal parameters.

The panel "Work with Entry fields" in the Generation of an Entry only allows option "7=Convert to valid ITP name" to be used. It does not allow any Field definitions to be created, changed, deleted, etcetera. The names initially used in the list of Fields are read from the file definition (DDS Name or TEXT)./ Refer to DID-source menu/Work with System values (page 41) for more information. When de DDS TEXT is used this is most likely not a valid ITP name. Before the field can be added to the Entry its name must first be changed to a valid ITP name. This can be done automatically by using the option "7=Convert to valid ITP name" or manually by just entering a correct name in the list. If the name is still not a valid ITP name when the function key F20 = Add fields to entry is pressed or the name is not unique within the Entry the "Work with Entry fields" will display an error for this. If the name was incorrect ITP will also convert it to a valid name when displaying the error. After function key F20 = Add fields to entry is pressed and an error is displayed the option "7=Convert to valid ITP name" is no longer present, and the only way to change a field name is by manually changing it in the list.

The panel "Work with Formal parameters" in the Generation of an Entry only allows option "7=Convert to valid ITP name" and more importantly the options "4=Drop key field" and "6=Retrieve key field". Option "7=Convert to valid ITP name" is rarely used because the name initially used is read from the Field definitions which already have valid ITP Names. The dropping of key fields plays a crucial role in combination with the attribute that determines whether an Entry is Singular or Plural.

If an Entry is generated using a uniquely keyed file the panel "Work with Formal parameters" will initially list all the key fields of the file. With a uniquely keyed file this would mean that the Entry is Singular. With the ability to drop Formal parameters the Entry can be made Plural. This is best explained by an example: Suppose the file RELATION is uniquely keyed by the key field ID:

File:	RELATION								
Keys:	ID		Id						
Fields:	TYPE		Relation		Type		Relations		Type
	ID				Id				Id

File:	RELATION								
	NAME		Singular		Name		Plural		Name
	.....				.....				.....

In the plural Entry Relations the key-field Id is dropped, thus resulting in an Entry with no Formal parameters that would retrieve all RELATION records from the database.

Note that it is only allowed to drop the last Formal parameters. Because a keyed database file is also sorted in the sequence determined by the keys, it could involve extensive programming to create the data retrieval program for such an Entry. ITP does not support this in *DBMS400 nor in the generation of data retrieval programs. Therefore, it is not possible to drop Formal parameters that are not the last Formal parameters of an Entry during the generation process.

> **Note**
> ITP does not support all DDS field types. Refer to Supported DDS field types (page 72) for more information.

Data retrieval

When the Entry is used in an ITP Model the data described by the Entry has to be retrieved from the database. ITP supports two interfaces to retrieve data using the ITP iSeries Connection:

1. *DBMS400: this can only be used when the Entry was generated; the ITP iSeries Connection itself retrieves the data from the database file the Entry was generated over.
2. Program interface; the ITP iSeries Connection calls the program registered in the Entry. The program is called using the library list of the host DataManager job in which it is called (refer to the "ITP Administration manual").

The function to be performed by both these interfaces is exactly the same:

- Key selection: if a Main entry has formal parameters, is used in a WITH or FORALL construction with an IN (as opposed to the PATH), and no key-values are passed to the ITP Model.
- Data retrieval: if an Entry is used in an ITP Model.

The Singular/Plural attribute of an Entry has no influence on the function performed for either the "key selection" or the "data retrieval". The Singular/Plural attribute only specifies how the Entry must be used in an ITP Model (in a WITH or a FORALL construct). The intended working of *DBMS400 or the program is always the same. Both the *DBMS400 interface and the program are called with two parameters for ITP internal use followed by a parameter for every Formal parameter of the Entry. The first two parameters are both defined as 10 bytes alphanumeric, the other parameters are passed conform the definitions of the appropriate Formal parameter.

Part of the Data retrieval interface is the way in which data is send to the ITP Model that uses the Entry. This is done by a call to the program ISNDDTA (since version 2.1) or the QSNDDTAQ program in the ITP iSeries Connection option "Pre-2.1 Connection legacy". The first 4 parameters of the programs are the same as the parameters of the iSeries program QSNDDTAQ (which was actually used in ITP/400):

Parameter 1 10 bytes ITP internal use; is equal to the first parameter of the retrieve

	alphanumeric	data interface.
Parameter 2	10 bytes alphanumeric	ITP internal use; is equal to the second parameter of the retrieve data interface.
Parameter 3	Packed decimal (5,0)	Number of bytes passed in parameter 4.
Parameter 4	Buffer with length indicated by parameter 3	A buffer containing the data of one record retrieved and to be returned to the ITP Model.

Intended working of Key selection

- Parameter 1 contains " KSTART " (space, KSTART, space, space, space)
- Parameter 2 contains a zoned (10,0) number indicating the number of records to be retrieved.
- Parameters 3 and further contain an actual value for the Formal parameters of the Entry
- Position in the database (file) using the values passed in the parameters 3 and higher.
- Read all records from the database (file) in sequence, but no more than the amount passed in the second parameter.
- Use a call to ITP's ISNDDTA program to return to ITP for every record that is read the key values and the values for any fields that are defined as Screen fields.

The key selection functionality is only required for Main entries, only when the Main Entry has Formal parameters defined, and only when these Main entries are used in a WITH or FORALL construct with an IN part, and no key values are specified when running the ITP Model. Refer to the ITP Integrator Manual for more information on how to specify the key values.

Intended working of Data retrieval

- Parameter 1 is ignored and can contain anything but " KSTART " (see Intended working of key selection)
- Parameter 2 is ignored.
- If applicable, parameters 3 and further contain an actual value for the Formal parameters of the Entry
- Position in the database (file) using the values passed in the parameters 3 and higher.
- Read all records from the database (file) that are identified by the values passed in the parameters 3 and higher (read by key).
- Return every record that is read to ITP using a call to ITP's ISNDDTA program

The data retrieval functionality is always required for an Entry.

DBMS400 versus Program interface

When *DBMS400 is used with an Entry, the ITP iSeries Connection will perform exactly the function described in the "Intended working of Key selection" and "Intended working of Data retrieval". An advantage of using *DBMS400 is that no separate program object is needed for an Entry. Disadvantages of *DBMS400 are that there is no way to change ore adapt the working of an Entry to fit some special needs and *DBMS400 can only be used with generated Entries.

The big advantage of the program interface is that the program can be changed to do virtually anything as long as the parameter and ISNDDTA interface requirements are satisfied. A disadvantage of the program interface is that the program and its source are two extra objects that

need to be maintained. The default libraries to store the program and its source in are the Object library and the Source library of the DID-source, refer to Work with DID-sources (page 34).

Generating data retrieval programs

When an Entry is generated one of the attributes of the Entry prompted on the "Generate Entry" display is the "PGM data retrieval". By default this attribute contains "*DBMS400". If a data retrieval program must be generated this value must be changed to the name of the program. Another possibility is to change this attribute after the Entry is generated (using option "2=Change" on the "Work with Entries (page 42)" panel.

The programs can be generated by using option "25=Generate Data retrieval" on the "Work with Entries (page 42)" panel. If Enter is pressed with option 25 then the default program generator is used with the default generation options. If option 25 is used in conjunction with the function key "F4=Prompt" the "Generate Program 'Data retrieval'" display is prompted where the program generator and other generation options can be changed. The default settings are determined by the choices registered with the "Work with User options GENPGMDTA (page 34)" from the "User options (page 34)" menu in "ITP management (page 33)".

There are currently three program generators present in the ITP/SDK for iSeries:

1. GENDTA.RPG Generator for RPG programs.
2. GENDTA.CBL Generator for COBOL programs.

> **Warning**
> Beware this generator does not include the functionality Key selection in the program.

3. GENDTA.C Generator for ILE C programs.

These generators are all programs located in the program library (page 17) of the SDK. The generators write the source of programs in a source physical file in the Source library of the current DID-source (page 34), and then call the appropriate compiler to create the programs in the Object library of the current DID-source (page 34).

The source physical files where the programs will be written to are:

1. GENDTA.RPG QRPGSRC
2. GENDTA.CBL QLBLSRC
3. GENDTA.C QCLESRC

The names of these source physical files cannot be changed. The files (together with the source physical file QDDSSRC) are automatically created in the Source library of the DID-source (page 34) when the DID-source is created.

> **Note**
> ITP does not guarantee correct and compilable programs to be generated. For instance in RPG a file to access cannot have a name with a length of more than eight characters, where DB2 UDB for iSeries does allow file names with a length of up to ten characters. If such a limit of the generators is encountered the source of the incorrect program must be manually changed and compiled to create the program.

After the source of the programs is generated the appropriate compiler is called to create the programs in the Object library of the current DID-source (page 34). The compilers are called by a CL program per generator with certain compiler options set explicitly, while other compiler options are default:

1.	GENDTA.RPG	DCRTRPGPGM	CRTRPGPGM with explicit options CVTOPT(*DATETIME *VARCHAR) ALWNULL(*YES)
2.	GENDTA.CBL	DCRTCBLPGM	CRTCBLPGM with explicit options CVTOPT(*VARCHAR *DATETIME)
3.	GENDTA.C	DCRTBNDC	CRTBNDC with explicit options OUTPUT(*PRINT) OPTIMIZE(*FULL) LANGLVL(*EXTENDED) REPLACE(*YES)

To change the compiler options or even the calling of the compiler itself, the sources of these CL programs may be retrieved and changed.

Do not change the program interface, make sure that the CL creates a callable program (no module or service program), and let the CL program be library list neutral (i.e. leave the library list as it was before calling the program).

Screen fields

ITP will display a Key selection window when running an ITP Model in which a main Entry is used that has Formal parameters for which no values are specified. ITP will at least display the actual values of the Formal parameters in this window. On the "Work with Entries (page 42)" panel, additional fields that will be displayed in the Key selection Window can be registered.

Before actually generating the data retrieval program or downloading the DID to the PC the Screen fields must be registered using option "36=Screen fields" on the "Work with Entries (page 42)" panel. Registering is simply adding a sequence number for the Field on the "Seq. numbers screen fields PGM key retrieval" panel.

The Key selection window that is displayed on the PC while running the ITP Model displays the actual values of the Formal parameters supplemented with the values of the registered Screen fields.

Regenerating Entries

When an Entry is created or generated the following dependencies (rows in the table) are important:

<table>
<tr><td>Database file</td><td>Entry</td><td>Data retrieval program (optional)</td></tr>
<tr><td>File name</td><td>File to access</td><td>File to access</td></tr>
<tr><td>Record format name</td><td>Record format to access</td><td>Record format to access</td></tr>
<tr><td rowspan="2">Key fields</td><td rowspan="2">Formal parameters</td><td>Program parameters</td></tr>
<tr><td rowspan="2">Key selection data returned with call to ISNDDTA</td></tr>
<tr><td rowspan="2">Fields</td><td>Screen fields</td></tr>
<tr><td>Entry fields</td><td rowspan="2">Result returned with call to ISNDDTA</td></tr>
<tr><td>Record length</td><td>Record Length</td></tr>
<tr><td></td><td>PGM data retrieval (optional)</td><td>Program name</td></tr>
</table>

If for instance a field definition in the database file changes, the corresponding Entry field must be changed accordingly, which might also require the data retrieval program to be changed. If the change of the field definition was only the change of its type without changing its length, then the program need not be changed if the program was a generated C-program because this references individual fields only type-independent. But if the program was a generated RPG or Cobol program it has to be changed depending on the exact type change.

It is not possible to give an exact recipe for what to do when something changes because the number of possible changes is too large. In most cases it would be much simpler and faster to recreate the Entry from scratch. This has one big disadvantage: Entries do not only exist in a DID-source; Entry-Subentry relations may be defined as well. If an Entry is deleted prior to recreation all Entry-Subentry relations the Entry played a role in are deleted as well. To prevent the loss of Entry-Subentry relations the possibility to re-generate Entries is available: options Option "61=Check re-generation of entry" and "62=Re-generate entry" on the panel Work with Entries (page 42).

Re-generation reads the current definition of the Entry and compares it to a complete recreation of the Entry. It tries to match field and formal parameter definitions and detects differences, keeping as much from the existing Entry (like field names) as possible. Option "61=Check re-generation of entry" does not change anything but presents a report of the differences and changes in a spool file, whereas option "62=Re-generate entry" actually changes the Entry.

Note that there are decisions the re-generation process cannot make. The re-generation process for instance has no means of deciding whether any of the original Formal parameters or Fields were dropped or deleted, or new Formal parameters or Fields were added. In this case the re-generation process assumes that new Fields were added, and they have to be dropped or deleted manually again after re-generation.

Another issue is when the DDS name of a Field in the database has changed. The re-generation process then decides that the Field with the old name was deleted and a new Field was added. This is especially important because Formal parameters and Fields play a role in the Entry-Subentry relations. Only where a Formal parameter or the Field used in an Actual parameter has not

changed the re-generation process can guarantee that the Entry-Subentry relations are still valid. It is therefore always a good idea to do a "Check DID-source" from the panel Work with DID-sources (page 34) after re-generating Entries. On the panel Work with Sub entries (page 58) there is also the possibility to do a "Check sub entry", and on the panel Work with Actual parameters (page 60) the function Synchronize exists.

Also note that the re-generation process doe not include the re-generation of any data retrieval programs; just use option "25=Generate Data retrieval" again.

Variable length fields

When Entries are (re-)generated for database files that contain variable length fields (DDS keyword VARLEN) manual changes to the Field definition of the Entry are needed.

The generation process defines the variable length field to be of a length that is equal to the maximum number of bytes that the field can occupy in the record. This includes the first 2 bytes that contain a binary number indicating the actual number of bytes in the field. Because these first 2 bytes contain a binary value, unpredictable results may occur when using the Field in an ITP Model. After (re-)generation of the Entry the definition of the variable length field should be changed to start on an offset that is 2 bytes further in the record and with a length that is 2 bytes less. It is also recommended to manually define an extra Field with a sequence number just before the variable length field, starting at the original offset of the variable length field, and defined as type binary with 2 bytes in length.

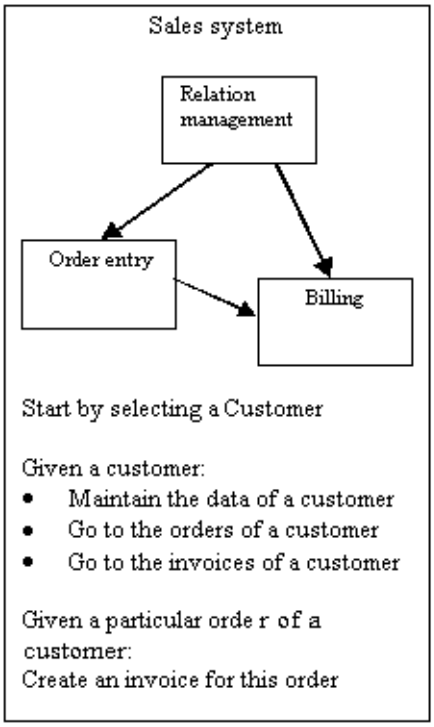
To illustrate the above:

Database file			Generated Entry Fields					Entry Fields after manual changes				
			Seq.	Field	Type	Off.	Length	Seq.	Field	Type	Off.	Length
FLDA	10A	TEXT ('Field A')	110	Field_A	T	123	10	110	Field_A	T	123	10
FLDB	100A	VARLEN TEXT ('Field B')	120	Field_B	T	133	102	119 120	Len_Field_B Field_B	B T	133 135	2 100
FLDC	10A	TEXT ('Field C')	130	Field_C	T	233	10	130	Field_C	T	233	10

DID Sources and modularity

Information systems can usually be subdivided into one or more logically and physically separated subsystems. The ITP/SDK for iSeries supports this kind of subdivision by the possibility to create a DID Source per subsystem. Multiple DID Sources can then be combined into one or more DIDs. The connections between those subsystems are described in the mechanism of importing and exporting Entries.

The concept of DID Sources and modularity, and the mechanism of import and export can best be introduced by the following example where the individual subsystems and DID Sources have no knowledge of any other subsystem or DID Source except for the interface. They have to connect to:



DID: Sales

Main Entry Customer	FROM RELATIONS
Main Entry Order	FROM ORDERS
Main Entry Invoice	FROM BILLING
DID - source: RELATIONS	
Entry Customer	Exported
Entry Address	
Import: Order	FROM ORDERS
Import Invoice	FROM BILLING
DID - source: ORDERS	
Entry: Order	Exported
Entry: Order details	
Import: Customer	FROM RELATIONS
Import: Order	FROM ORDERS
DID - source: BILLING	
Entry: Invoice	Exported
Entry: Price list	
Import: Customer	FROM RELATIONS

With the DID Sales in this example ITP Models can be build that:

- Create a document with the data of the customer
- Create an overview of the orders listed for the customer
- Create invoices and reminders for the customer
- Create an invoice for a particular order
- Create an overview of all non-satisfied invoices for the customer
- Create an overview of all non-satisfied invoices in the system

In the SDK Entries are by default always local to the DID-source they are defined in. To be able to import an Entry in another DID Source the Entry must first be explicitly tagged as "export allowed" by using option "15=Export" on the panel Work with Entries (page 42) or by manually setting the export attribute of the Entry.

To actually import an Entry from another DID Source the function key F22 = Import must be used on the panel Work with Entries (page 42). This function then prompts the panel Import Entry on which the name of the Entry to import and the DID Source it is to be imported from must be entered. It is possible to use function key F4 = Prompt on the panel Import Entry to first select a DID
Source from all DID Sources registered in the SDK and then select an Entry from all the Entries tagged as "export allowed" in that DID Source.

Note
Entries can be imported from any DID Source registered in the ITP/SDK for iSeries even if they are not linked to the same DID the DID Source that does the import is linked to. A link between the DID and a DID Source can be deleted even when there are still Entries imported from that DID Source into another DID Source still linked to the DID.

The fact that Entries are used in a DID that are imported from a DID Source not linked to the DID is not detected unless a "Check DID" is done on the DID from the panel Work with DIDs (page 18).

Imported Entries are the same as Entries defined in the DID-source itself with some restrictions:

- Imported Entries may not be registered as Main entries (and thus changing the Screen field definitions is not allowed).
- It is not possible to create Subentries for the Imported Entry.
- Attributes of imported Entries cannot be changed, and thus they cannot be exported and no offsets may be recalculated.
- Imported Entries cannot be copied or re-generated.
- No data retrieval program may be generated for the imported Entry.
- It is not possible to work with the Field definitions of the imported Entry.

It is allowed to change the Formal parameters of an Imported Entry but this is something that generally should not be done! The only reason to allow this is in case the Formal parameters of the original exported Entry are changed to prevent a required delete and re-import of the Entry. Multiple Aliases that are used many times as Subentries could be defined for the imported Entry, and there is no function to synchronize imported Entries in the SDK. So to prevent a lot of work one could simply change the Formal parameters of the imported Entry according to the changes of the original exported Entry.

Work with Sub entries

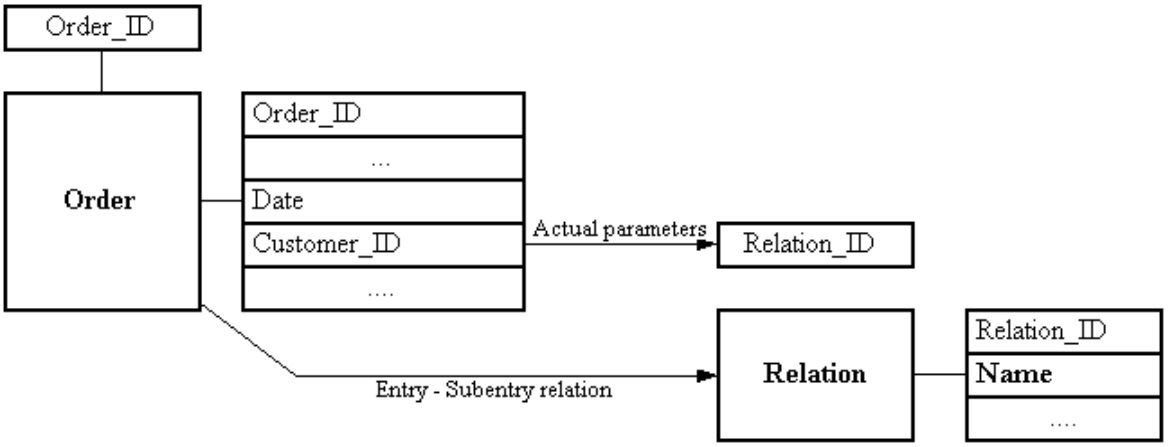
In relational databases it is very common that relations exist between the records in different files. A file on itself is often no more than a means of storing a lot of records of the same type, while individual records may have a relation to records in other (or even the same) file. Suppose the following example where there are two files involved: "Orders" and "Relations":

Orders						Relations		
Order ID		Date	Customer ID	…..		Relation ID	Name	…..
…..		…..	…..	…..		…..	…..	…..
23473856		20020112	237648454	…..		237648454	Acme Inc.	…..
23473857		20020112	948534548	…..		…..	…..	…..
…..		…..	…..	…..		948534548	Insurance Inc.	…..
						…..	…..	…..

In this example the connection between a record from the "Orders" file and a record from the "Relations" file is by the "Relation- IDs" stored in the "Customer ID" fields of the records of the "Orders" file. Given a particular record from the "Orders" file the customer that placed the order is identified by the "Relation ID" stored in the" Customer ID" field of this record.

When describing the example in the ITP/SDK for iSeries one would generate two Entries; Order

and Relation. To indicate the connection between Order and Relation, Relation must be registered as a Subentry of Order. To complete the Subentry definition the Field to be used to make the connection (Customer ID from the Entry Order) must be defined as the Actual parameter in the Entry Subentry relation.



Using option "17=Sub entries" with an Entry on the panel Work with Entries (page 42) displays the list of all Subentries of the Entry on the panel Work with Sub entries.

Function keys

F6=Create

Register a new Subentry with the Entry.

Options

4=Delete

Delete the registration of Subentry with the Entry (including the Actual parameters).

5=Display

Display the primary attributes of the Subentry.

This exactly the same function as when option "5=Display" is used on the "Work with Entries (page 42)" panels with the Entry that is registered as Subentry

18=Actual parameters

The list of all Actual parameter definitions is prompted on the "Work with Actual parameters".

21=Check sub entry

Checks if the definition of Formal parameters of the Subentry are consistent with the Actual parameters specified.

An Entry can only be listed once as a Subentry of another Entry. If more than one relation between two Entries exist Aliases should be used to define the role of an Entry in the relation. These Aliases can then be registered as the Subentries. Refer to Work with Aliases of entry (page 58) for more

information.

Work with Actual parameters

After registering an Entry as Subentry the values actually used to make the connection between the Entry and Subentry must be filled in. These values are called Actual parameters. When a Subentry is registered, the Formal parameters of this Entry are used to build an empty list of Actual parameters. Using option "18=Actual parameters" with a Subentry on the "Work with Sub entries" panel displays the list of Actual parameters on the "Work with Actual parameters" panel.

Function keys

F6=Synchronize

Rebuilds the list of Actual parameters according to the Formal parameter definitions of the Subentry. If a Formal parameter is no longer part of the Entry it is deleted from the list. New Formal parameters defined with the Subentry are added to the list.

Options

2=Change

Changes the list item. Use this option to enter the Actual parameter.

5=Display

Displays the attributes of the Actual parameter. These are the attributes of the corresponding Formal parameter of the Subentry and the attributes of the Actual parameter used.

As an Actual parameter either a Field from the Entry or a constant value (but not both) can be used. This can be entered on the "Create Actual parameter" panel that will be prompted when option "2=Change" is used on the "Work with Actual parameters" panel. It is not possible to enter a Field from the Entry as Actual parameter if the definition of this Fields does not match the definition of the Formal parameter. For Constant values there is no check done if the value entered is consistent with the definition of the Formal parameter of the Subentry. The only check performed is whether the constant value has to be numerical or not.

Using function key "F4=Prompt" on the "Create Actual parameter" panel displays a list of all Fields from the Entry with a definition consistent with the definition of the corresponding Formal parameter of the Subentry. On this "Select Actual parameter" panel one of the listed fields can be selected as Actual parameter.

Constant values can be entered in the input field "Constant value". They have to be entered directly without delimiters and can be no more than 25 characters in length. If more or less is entered than required by Formal parameter the behavior of ITP is unspecified.

Work with Entry fields

Using option "16=Fields" with an Entry on the "Work with Entries (page 42)" panel displays the list of all Fields of the Entry on the "Work with Entry fields" panel. If the Entry was generated this list was automatically created in the generation process. If the Entry is created manually this list is initially empty. The Fields of an Entry is the information that is returned to the ITP Model when

the Entry is used. The ITP Model uses the names defined here to individually address the information.

There are two versions of the "Work with Entry fields" panel: the "normal" one that will be displayed when an Entry is already created and a special one that is displayed as a part of the Entry generation process (page 46).

On the special "Work with Entry fields" only the name of a Field can be changed and only option "7=Convert to valid ITP name" is available. On the normal "Work with Entry fields" panel the Fields of the Entry can be manipulated:

Functions keys

F6=Create

Create a new Field for the Entry.

Options

2=Change

Change the attributes of a Field for the Entry.

3=Copy

Copy the Field to create a new Field for the Entry.

4=Delete

Delete the Field from the Entry.

5=Display

Display the attributes of the Field.

The Field-information consists of:

- the primary Field attributes (the name of the Field, the sequence number of the Field, etc.);
- the extended Field attributes (the DDS name of the Field, the offset of the Field in the database file, etc.)

The primary attributes of a Field are:

Field name A case sensitive name of maximal 30 characters that serves as a unique identification of the Field inside the Entry. This name must start with an uppercase and may not contain any blanks.

Sequence number A five-digit number that uniquely determines the position of the Field in the Entry.

Offset When an ITP Model uses the Entry all Fields of the Entry are returned in one block of data to the Model. The offset determines where in this block of data the information of the Field starts, relative to the start of

	the block (thus the smallest possible offset is 0).
Data type	The data type of the Field. This can be TEXT (T), PACKED (P), ZONED (Z), BINARY (B) or Unicode (W). ITP does not support all DDS field types: refer to "Supported DDS field types (page 72)" for more information.
Length	The length in bytes of the Field.
Decimal positions	Number of decimal positions in the Field. Only used if the data type of the Field is numeric (PACKED, ZONED or BINARY)
Alignment	Alignment of the Field. This can be either left aligned (L) or right aligned (R). Only used if the data type of the field is not numeric (TEXT or Unicode)
Field generation	Can be either "Automatically" (A), "Manually" (M) or "Automatically but changed" (blank). This tag indicates the way in which the Field was created.

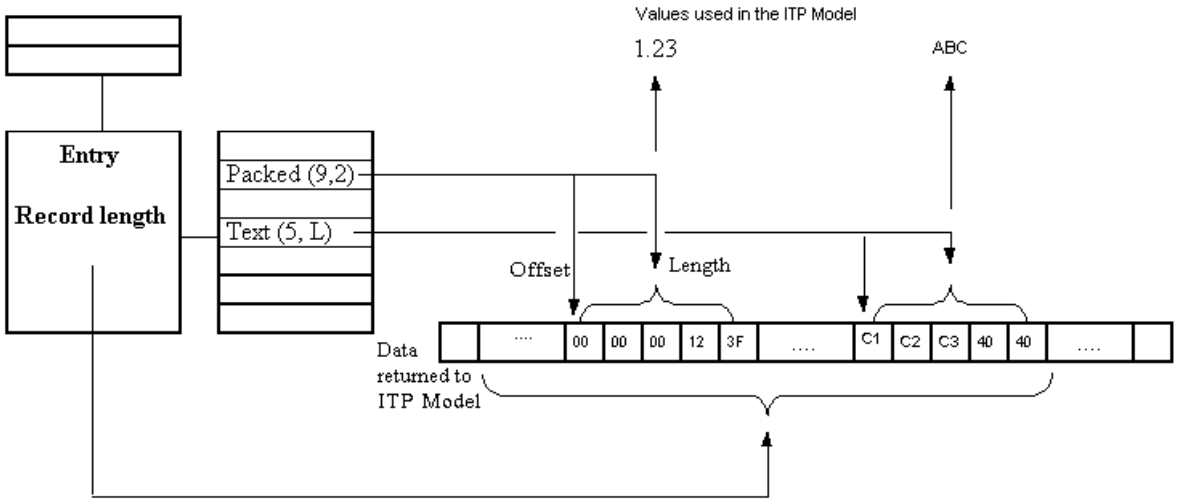
The extended attributes of a Field are automatically filled in when an Entry is generated and contain extra information from the File definition. The extended attributes are not used when a Field is created manually. The extended attributes are:

File field name (DDS name)	The original DDS name of the Field in the database file the Entry was generated over.
Sequence number file field	The original sequence number of the Field in the database file the Entry was generated over.
Offset file field	The original offset of the Field in the record of the database file the Entry was generated over. The smallest offset possible is 1 to indicate the Field started on the first position of the record.
Sequence number of key field	If the database file field was a key field in the database file the Entry was generated over, this sequence number indicates its position relative to the other key fields in the file. 0 indicates that the database file field was not a key field.

Entry Creation and Generation (page 46) contains more information on the relation between the Field definitions and the database file definitions that an Entry can be created over and about the data retrieval mechanism ITP uses. From the point of view of an ITP Model, Fields are the objects that contain the information from the database: it is by using a Field in the ITP Model that the content of the field is used. From the point of view of the technique of retrieving data from the database, Fields only determine how the ITP Model must interpret the data that was returned to the ITP Model.

The data retrieval mechanism returns a block of data. The record length attribute of the Entry determines the size in bytes the block of data is supposed to have. After the ITP Model receives the block of data and a Field is used in the Model the data is interpreted using the attributes of the Field. Only then the ITP Model takes a "Field length" number of bytes from the block of data returned starting at the "Field offset" and translates it according to the "Field type" attributes of the

Field.



The record length attribute of the Entry is the most important in the data retrieval mechanism. On Entry generation this attribute is retrieved from the record length of the file definition the Entry was generated over. When the Entry is manually created or Field definitions of the Entry are changed after generation, the record length should reflect the Field definitions of the Entry. The exact formula is: record length = (offset of the last field + length of the last field). In general it is possible to have bytes returned to the ITP Model that are not used in any Field of the Entry. Option "19=Calc. Offsets" on the "Work with Entries (page 42)" may be used to calculate the record length of the Entry and the offsets of the Fields based on the sequence numbers and the length of the field. This option assumes that there are no unused bytes in the record, and moves the fields by adjusting the offsets of the fields so that they start directly behind the previous Field.

Fields as actual parameters for Subentries

Fields can be used as Actual parameters (page 60) in Entry Subentry relations. Beware that changes to Entry Fields may require that Actual parameters of already existing Subentries must be changed as well.

Unicode

The last couple of years character encoding schemes like Unicode have increased in popularity. To support Unicode (as UCS-2 data) the Unicode data type (W) was introduced in version 2.1 of the ITP/SDK for iSeries.

The ITP/SDK for iSeries does not automatically differentiate between several character data encoding schemes. When (re-) generating an Entry, Field types are automatically set to type Text (T). When the Field contains UCS-2 data the Field type must always be changed manually to Unicode (W) to let ITP know how to interpret the bytes correctly when they are returned to the ITP Model.

An important restriction of the Unicode data type in the ITP/SDK for iSeries is that the length of the Field must be an even number of bytes, because UCS-2 is a double byte character set. The order of the two bytes must be the native iSeries order. This means that "normal" characters like the "A" are represented with 0 in the first byte and 65 in the second byte.

Work with Formal parameters

Using option "18=Formals" with an Entry on the panel Work with Entries (page 42) displays the list of all Formal parameters of the Entry on the panel Work with Formal parameters. If the Entry was generated this list was created in the generation process. If the Entry is created manually this list is initially empty. The Formal parameters of an Entry are the information that is needed to retrieve the intended records when the Entry is used.

There are two versions of the panel "Work with Formal parameters"; the 'normal' one that will be displayed when an Entry is already created and a special one that is displayed as a part of the Entry generation process (page 50).

On the special "Work with Formal parameters" only the name of a Formal parameter can be changed and Formal parameters can be dropped or retrieved when they were previously dropped. Only options "4=Drop key field", "6=Retrieve key fields" and "7=Convert to valid ITP name" are available. On the normal "Work with Formal parameters" panel the Formal parameters of the Entry can be manipulated.

Function keys

F6=Create

Create a new Formal parameter for the Entry.

Options

2=Change

Change the attributes of a Formal parameter for the Entry.

3=Copy

Copy the Formal parameter to create a new Formal parameter for the Entry.

4=Delete

Delete the Formal parameter from the Entry.

5=Display

Display the attributes of the Formal parameter.

The Formal parameter-information consists of:

- the primary Formal parameter attributes (the name of the Formal parameter, the sequence number of the Formal parameter, etc.);
- the extended Formal parameter attributes (the DDS name of the Formal parameter, the offset of the Formal parameter in the database file, etc.)

The primary attributes of a Formal parameter are:

Formal parameter name	A case sensitive name of maximal 30 characters that serves as a unique identification of the Formal parameter inside the Entry. This name

must start with an uppercase and may not contain any blanks.

Sequence number A five-digit number that uniquely determines the position of the Formal parameter in the Entry.

Data type The data type of the Formal parameter. This can be TEXT (T), PACKED (P), ZONED (Z), BINARY (B) or Unicode (W). ITP does not support all DDS field types: refer to "Supported DDS field types (page 72)" for more information.

Length The length in bytes of the Formal parameter.

Decimal positions Number of decimal positions in the Formal parameter. Only used if the data type of the Formal parameter is numeric (PACKED, ZONED or BINARY)

Alignment Alignment of the Formal parameter. This can be either left aligned (L) or right aligned (R). Only used if the data type of the Formal parameter is not numeric (TEXT or Unicode)

Parameter generation Can be either "Automatically" (A), "Manually" (M) or "Automatically but changed" (blank). This tag indicates the way in which the Formal parameter was created.

The extended attributes of a Formal parameter are automatically filled in when an Entry is generated and contain extra information from the File definition. The extended attributes are not used when a Formal parameter is created manually. The extended attributes are:

File field name (DDS name) The original DDS name of the Formal parameter in the database file the Entry was generated over.

Sequence number file field The original sequence number of the Formal parameter in the database file the Entry was generated over.

Offset file field The original offset of the Formal parameter in the record of the database file the Entry was generated over. The smallest offset possible is 1 to indicate the Formal parameter started on the first position of the record.

Sequence number of key field If the database file field was a key field in the database file the Entry was generated over, this sequence number indicates its position relative to the other key fields in the file. 0 indicates that the database file field was not a key field.

Entry Creation and Generation (page 46) contains more information on the relation between Formal parameter definitions and the database file definitions an Entry can be created over and about the data retrieval mechanism ITP uses. Formal parameters are the objects in which the information has to be stored that enables the data retrieval mechanism to retrieve the intended data for the Entry. The use of the Formal parameters is usually hidden in the ITP Model and only visible when the PAR- instructions are used in a WITH or FORALL construct.

Formal parameters have exactly the same attributes as Entry Fields (page 60), but only a few of these attributes are really important for Formal parameters. The actual values the ITP Model passes to the data retrieval mechanism are passed individually for every Formal parameter. The sequence number of the Formal parameter determines the order in which the individual values are passed. The type (Data type, Length, Decimal positions, Alignment) of the Formal parameter determines:

- how a value from the ITP Model must be passed to the data retrieval mechanism if a PAR instruction is used in a WITH or FORALL construct in the Model;
- which Fields may be used as Actual parameters (page 60) when the Entry is used as a Subentry.

Beware that if Formal parameters are changed:

- any already existing Subentry relation in which the Entry is used as a Subentry (page 58) has to be changed accordingly;
- the parameters of a data retrieval program (page 46) for the Entry must change as well;
- *DBMS400 may no longer be used as a data retrieval mechanism (page 46) if the change is other than the dropping of a Formal parameter or the retrieving of a previously dropped Formal parameter.

Unicode

The last couple of years character encoding schemes like Unicode have increased in popularity. To support Unicode (as UCS-2 data) the Unicode data type (W) was introduced in version 2.1 of the ITP/SDK for iSeries.

The ITP/SDK for iSeries does not automatically differentiate between several character data encoding schemes. When (re-) generating an Entry, Formal parameter types are automatically set to type Text (T). When the Formal parameter is intended to contain UCS-2 data the Formal parameter type must always be changed manually to Unicode (W) to let ITP know that Unicode characters have to be passed to the data retrieval mechanism.

An important restriction of the Unicode data type in the ITP/SDK for iSeries is that the length of the Formal parameter must be an even number of bytes, because UCS-2 is a double byte character set. The order of the two bytes must be the native iSeries order. This means that "normal" characters like the "A" are represented with 0 in the first byte and 65 in the second byte.

Work with Aliases of entry

In relational databases it is very common that relations exist between files. A file is often no more than a means of storing a lot of records of the same type, while the exact relation between individual records is determined in a different way. Suppose the following example where there are two files involved: "Orders" and "Relations":

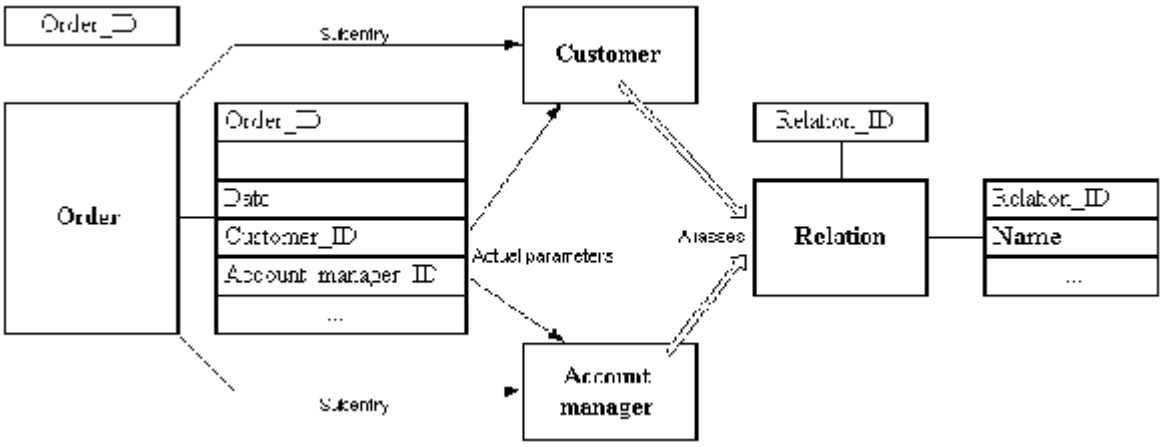
Orders							**Relations**		
Order ID		Date	Customer ID	Account manager ID	…..		Relation ID	Name	…..
…..		…..	…..	…..	…..		…..	…..	…..
23473856		20020112	237648454	789563478	…..		237648454	Acme Inc.	…..

Orders							Relations		
23473857		20020112	948534548	457769767	…..		…..	…..	…..
…..		…..	…..	…..	…..		457769767	Smith	…..
							…..	…..	…..
							789563478	Jones	…..
							…..	…..	…..
							948534548	Insurance Inc.	…..
							…..	…..	…..

In this example the file "Relations" contains information on all relations, e.g., Companies, Organizations, Persons. The connection between the file "Orders" file and the "Relations" is by the "Relation- IDs" stored in the "Orders" file. But there is a paramount distinction between the connections indicated by the "Relation- IDs" in the column: "Customer IDs" and those in the column "Account manager IDs": the relations indicated in these columns play a different role in the particular order. Those in the column: "Customer IDs" indicate the customer that placed the order; those in the column "Account manager IDs" indicate the employee that is responsible for the order.

When describing the example in the ITP/SDK for iSeries one would generate two Entries: Order and Relation. To indicate the connection between Order and Relation one would like to register Relation twice as a Subentry of the Entry Order: one with the Field Customer_ID and one with the Field Account_manager_ID as Actual parameter. However, this is not possible because an Entry can only be registered once in the Subentry list of another Entry. Even if this restriction did not exist it would be confusing to have two Subentries to the Entry Order that are both named Person: which Person would indicate the Customer and which the Account manager?

To solve this problem ITP has the concept of Aliases. An Alias entry is nothing more than another name for an existing Entry, where the name indicates a role the original Entry can play in respect to other Entries. Following the example of Orders and Relations this would lead to:



Alias Entries are the same as Entries defined in the DID-source, but: the only thing that an Alias Entry can be used for is as a Subentry with another Entry.

- Alias entries may not be registered as Main entries (and thus changing the Screen field

definitions is not allowed);
- It is not possible to create Subentries for the Alias entry;
- Attributes of Alias entries cannot be changed (and thus they cannot be exported and no offsets may be recalculated);
- Alias Entries cannot be copied or re-generated;
- No data retrieval program may be generated for the Alias entry;
- No Field definitions or Formal parameters may be changed.

Using option "14=Aliases" with an Entry on the panel Work with Entries displays the list of all Aliases of the Entry on the panel "Work with Aliases of entry".

Create a new Alias for the Entry.

Delete the Alias Entry. A window with a warning and an extra request for confirmation is displayed if the Entry is still used as a Subentry.

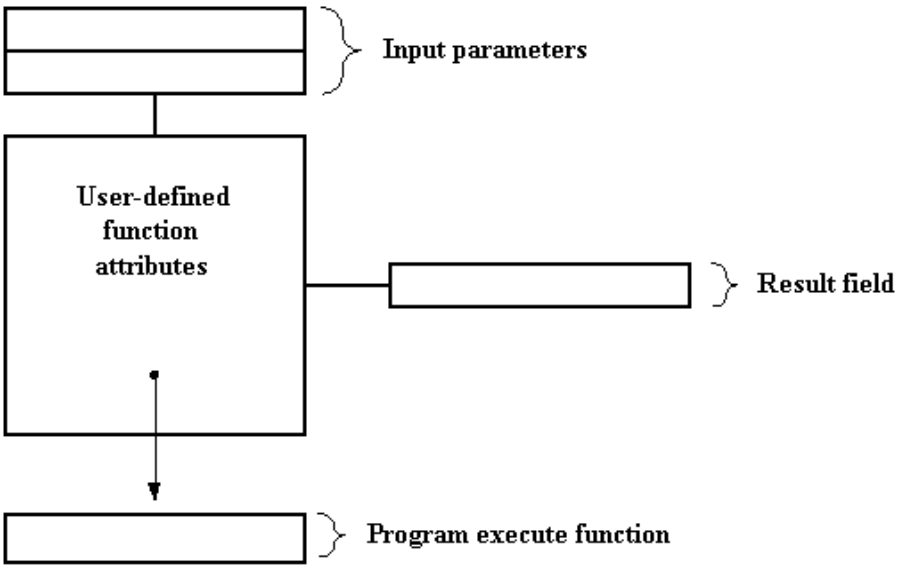
This is exactly the same function as when option "4=Delete" is used on the "Work with Entries" panels for the Alias Entry

Display the primary attributes of the Alias Entry.

This is exactly the same function as when option "5=Display" is used on the "Work with Entries" panels for the Alias Entry

Work with user defined functions

User-defined functions (often referred to as DID-defined functions) may be considered to be a special, restricted form of Entries (page 8). In fact they are basically the same as Entries in almost any aspect. The big difference is that User-defined functions can only return one Record with exactly one Result field.



A User-defined function can best be looked at as the description of the interface of a program (the program execute function). The Input parameters describe the parameters of the program; only input parameters are allowed. The Result field describes the data the program must send back to the ITP Model when it uses the function.

Another major difference between Entries and User-defined functions is that there is no automatic support for creating a User-defined function; everything has to be created manually. User-defined functions are intended to describe a particular kind of service that has to be performed during the ITP Model execution, without the necessity to write down a complete WITH or FORALL construct in the ITP Model document: User defined functions can be used like any other function in the ITP Model development language.

A User-defined function consists of:

- the User-defined function attributes, e.g. the name of the User-defined function, or a descriptive text
- the set of Input parameters
- the Result field
- a reference to an iSeries program

From the panel "Work with User defined functions" User-defined functions can be created, changed, deleted, etcetera.

Function keys

F6=Create

Create a new User-defined function record and set its attributes.

Options

2=Change

Change the attributes of the User-defined function.

3=Copy

Copy a User-defined function including its Input parameters and Result. The copy action is restricted to the same DID-source. Any referenced program execute function is not copied.

"4=Delete

Delete a User-defined function including its Input parameters and Result. Any referenced program execute function is not copied.

5=Display

Display the attributes of the User-defined function.

18=Input parameters

Displays the panel Work with Input parameters (page 71) that lists all Input parameters of the User-defined function.

19=Result

Displays the panel Work with Result (page 72) that lists the Result of the User-defined function.

27=Work with Object

Starts WRKOBJPDM for the program execute function object in the Object library of the DID-source.

30=Linked DIDs

Displays a window listing all DIDs where the User-defined function can be used.

The attributes of a User-defined function are:

User-defined function name	A case sensitive name of maximal 30 characters that serves as a unique identification of the User-defined function inside the DID-source. This name must start with an uppercase and may not contain any blanks.
Description	A descriptive text
PGM function execution	The name of an iSeries program. The program is called when the ITP Model uses the User-defined function and the program must send the Result back to ITP.

Program execute function

When the User-defined function is used in an ITP Model the ITP iSeries Connection will call the program registered in the User-defined function using the library list of the host DataManager job in which it is called, refer to the ITP Administration Manual. The program is called with two parameters for ITP internal use followed by a parameter for every Input parameter of the User-defined function. The first two parameters are both defined as ten bytes alphanumeric, the other parameters are passed conform the definitions of the appropriate Input parameter.

The only requirement on what the program has to do is that it must send the Result back to ITP using a call to the program ISNDDTA (since version 2.1) or the QSNDDTAQ program in the ITP iSeries Connection option "Pre-2.1 Connection legacy". The first four parameters of the programs are the same as the parameters of the iSeries program QSNDDTAQ, which was actually used in ITP/400:

Parameter 1	10 bytes alphanumeric	ITP internal use; is equal to the first parameter of the program execute function interface.
Parameter 2	10 bytes alphanumeric	ITP internal use; is equal to the second parameter of the program execute function interface.
Parameter 3	Packed decimal (5,0)	Length of the Result passed in parameter 4.
Parameter 4	Buffer with length indicated by parameter 3	A buffer containing the Result to be returned to the ITP Model.

Work with Input parameters

Using option "18=Input parameters" with a User-defined function on the "Work with User-defined function (page 39)" panel displays the list of all Input parameters of the User-defined function on the "Work with Input parameters" panel. After the User-defined function is created this list is initially empty. The Input parameters of a User-defined function are input parameters (preceded by the two input parameters for ITP internal use) of the program execute function (page 70) registered with the User-defined function.

On the "Work with Input parameters" panel the Input parameters of the User-defined function can be manipulated:

Function keys

F6=Create

Create a new Input parameter for the User-defined function.

Options

2=Change

Change the attributes of an Input parameter for the User-defined function.

3=Copy

Copy the Input parameter to create a new Input parameter for the User-defined function.

4=Delete

Delete the Input parameter from the User-defined function.

5=Display

Display the attributes of the Input parameter

The attributes of an Input parameter are:

Input parameter name	A case sensitive name of maximal 30 characters that serves as a unique identification of the Input parameter inside the User-defined function. This name must start with an uppercase and may not contain any blanks.
Sequence number	A five-digit number that uniquely determines the position of the Input parameter in the User-defined function.
Data type	The data type of the Input parameter. This can be TEXT (T), PACKED (P), ZONED (Z), BINARY (B) or Unicode (W). ITP does not support all DDS field types: refer to "Supported DDS field types (page 72)" for more information.
Length	The length in bytes of the Input parameter.

Decimal positions	Number of decimal positions in the Input parameter. Only used if the data type of the Input parameter is numeric (PACKED, ZONED or BINARY)
Alignment	Alignment of the Input parameter. This can be either left aligned (L) or right aligned (R). Only used if the data type of the Input parameter is not numeric (TEXT or Unicode)

Work with Result

Using option "19=Result" with a User-defined function on the "Work with User-defined function (page 39)" panel displays a list containing the Result of the User-defined function on the "Work with Result" panel. After the User-defined function is created this list is automatically created with an empty Result. The Result of a User-defined function is the information the program execute function (page 70) registered with the User-defined function has to send back to the ITP Model when the User-defined function is used.

On the "Work with Result" panel the Result of the User-defined function can be manipulated:

Supported DDS field types

The ITP/SDK for iSeries does not support all field types supported by DB2 UDB for iSeries. The following view lists the possible DDS field types and how ITP supports them:

Packed Decimal	(P)	Supported by ITP	
Zoned Decimal	(S)	Supported by ITP	
Binary	(B)	Supported by ITP	
Floating Point	(F)	Not supported by ITP	
Character	(A)	Supported as ITP Text Field	Variable length fields (VARLEN) are not supported as Formal parameter (1)
Date	(L)	Supported as ITP Text Field	Not supported as Formal parameter (1)
Time	(T)	Supported as ITP Text Field	Not supported as Formal parameter (1)
Time stamp	(Z)	Supported as ITP Text Field	Not supported as Formal parameter (1)
Hexadecimal	(H)	Supported as ITP Text Field	Not supported as Formal parameter (1)

DBCS-Only	(J)	Not supported by ITP
DBCS-Either	(E)	Not supported by ITP
DBCS-Open	(O)	Not supported by ITP
DBCS-Graphic	(G)	Not supported by ITP

(1) This means that *DBMS400 cannot handle Formal parameters of this type nor can the data retrieval programs generated with the SDK. Building a data retrieval program manually is the only option here.

Copy-APIs

Normally, working with the ITP/SDK for iSeries starts with the STRITP command. All SDK functions are performed inside of the context of the SDK. Exceptions to this rule are the ITP/SDK for iSeries APIs. These are commands that may be called outside of the context of the SDK, but perform functions that work with and possibly change the contents of the (data in the) SDK.

The supported APIs are:

COPYDID	Copy a DID from one iSeries library to another. Only the information of the DID and all DID-sources linked to it are copied. iSeries objects referenced in the DID are not copied.
COPYDIDINF	Copy a DID from one iSeries library to another. Only the information of the DID not including the information of DID-sources linked to it is copied.
COPYDIDSRC	Copy a DID-source from one iSeries library to another. Only the information of the DID-source is copied. iSeries objects referenced in the DID are not copied.

These APIs copy the information from one iSeries library to another (these libraries may not be the same) and operate directly on the physical files from the SDK database. The default values for the FROM parameters of these commands (FROMDIDLIB, FROMDIDFIL, FROMSRCLIB and FROMSRCFIL) are the names of the library and files for the standard SDK database. If any of the files to copy the information to does not exist, the COPY-API will create it.

The REPLACEDB-parameter of the command controls the action to take when one or more of the files to copy to already exist.

*NONE	The copy action will be cancelled. No copy will be performed.
*ADD	The information to be copied will be added to the information already present in the files to copy to.
*REPLACE	The information to be copied will replace the information already present in the files to copy to; the contents of the files that already exist will be cleared before the new information is written to it. Beware: it is not recommended to use this setting when the target database is the SDK database.

When *ADD is used in the REPLACEDB parameter of the Copy-API, an additional REPLACE-parameter on the command (REPLACEDID or REPLACESRC) controls what to do when the copy action would lead to duplicate information in the files to copy to (i.e. if the DID or DID-sources to copy are already present in these files).

*NONE	The copy action will be cancelled. No copy will be performed.
*REPLACE	The information to be copied will replace the information already present in the files to copy to; any information on the DID or DID-sources to copy that is already present in the files will be deleted before the new information is

written to it. Any other information (not related to the DID or DID-sources to copy) is not affected by the copy operation. Beware: this also applies to DID-sources linked to a DID to copy.

The official interfaces for these Copy-APIs are the commands. However, the sources of the commands and their command processing (CL) programs are part of the SDK distribution as well in the source physical files QCMDSRC and QCLSRC in the SDK program library.

Index