

**ITP**

# **ITP and ODBC**

**2003-11-21**

*This white paper describes the integration of ITP and ODBC to create advanced business document production solutions.*



## **INTELLIGENT TEXT PROCESSING**

## Table of contents

Table of contents .....	1
1. Introduction .....	2
2. Introduction to ITP .....	3
2.1 The architecture of ITP .....	3
2.2 Database abstraction .....	4
2.3 Model documents, models and result documents.....	4
2.4 The ITP document development process.....	5
2.5 Producing result documents.....	5
2.6 Printing ITP output .....	5
2.7 The ITP product family .....	6
3. The ITP Instruction language.....	7
4. ITP and ODBC .....	11
4.1 Database access .....	11
4.1.1 Entries .....	11
4.2 Integration .....	12

# 1. Introduction

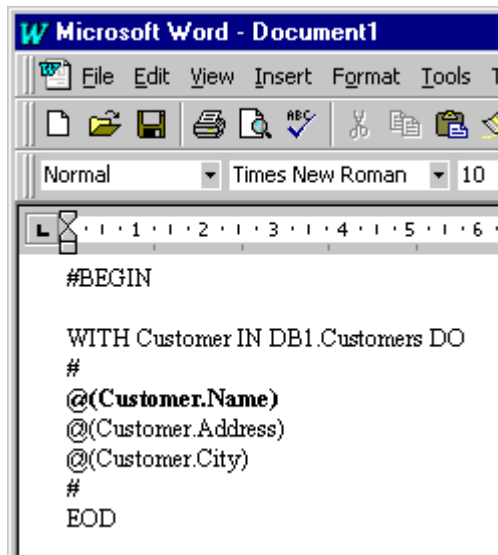
This white paper describes the ITP product and the integration between ITP and ODBC.

ITP is a client/server software solution that offers unprecedented flexibility in creating text/data merge applications for modern word processors, like Microsoft® Word and OpenOffice.org Writer. ITP connects these word processors to your databases. It does so by adding a flexible and easy-to-use instruction language to your word processor of choice.

ITP supports access to multiple database types. These types of access are called ITP Connections. Currently ITP supports access to Lotus Notes/Domino databases, AS/400 databases, Oracle databases, ODBC enabled databases, Mainframe databases and XML data. It is also possible to collect XML data via the HTTP protocol.

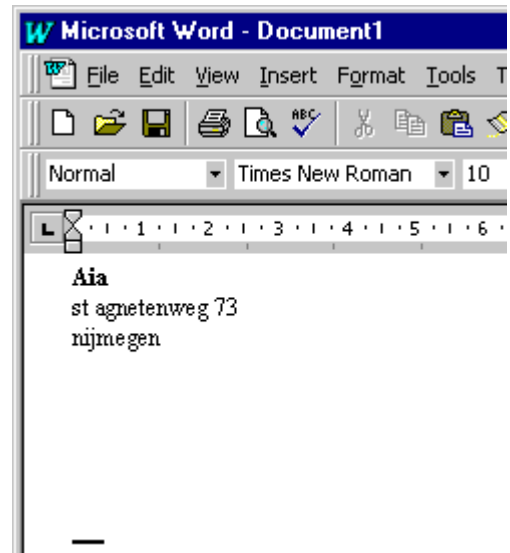
This white paper focuses on ITP's access to ODBC data. We start with an introduction to ITP and to ITP document development. We then move on to ODBC database access for ITP.

# White paper



A screenshot of the Microsoft Word interface showing a document with ITP merge code. The code is as follows:

```
#BEGIN  
  
WITH Customer IN DB1.Customers DO  
#  
  @(Customer.Name)  
  @(Customer.Address)  
  @(Customer.City)  
#  
EOD
```



A screenshot of the Microsoft Word interface showing the result of the ITP merge. The text in the document is:

```
Aia  
st agnetenweg 73  
nijmegen
```

## 2. Introduction to ITP

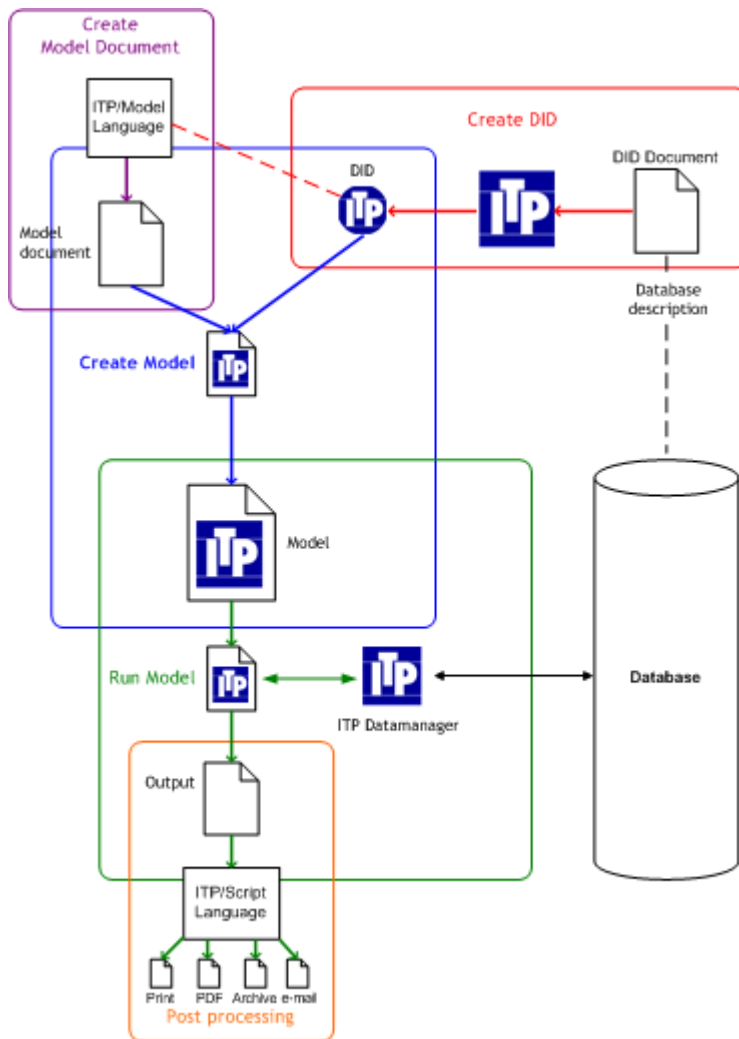
With ITP the model documents (merge documents) are developed in the word processor of your choice. After development of a model document, ITP creates the final output document(s) based on this model document. The word processor itself is not used during this creation process.

ITP combines all familiar word processor facilities, both textual as well as layout, in a seamless way with data from your corporate databases. Text and layout in the documents can be conditioned based on the data from the database. ITP accesses the database in real-time, so no downloading, pre-processing or copying of data is required.

Document production can be document driven, where the output is based on conditions and calculations coded in your document, that may or may not use database access. It can also be data driven, where the output is based on the contents of the database. If necessary, the document production or parts of it can be controlled by user interaction.

Very different types of documents can be developed with ITP, ranging from correspondence, invoices and quotations to policies, contracts and financial reports.

### 2.1 The architecture of ITP



This figure describes all basic ITP processes:

## ITP and ODBC

1. Create DID: A description of the database is created, the DID. The DID describes how to access the database using for example SQL queries or external data retrieval programs.
2. Create Model Document: Model documents are developed in the word processor of your choice. These documents contain ITP instructions with references to the ITP database description. Model documents are compiled to ITP Models.
3. Run Model: ITP Models are invoked to produce the actual result documents. During this process data are accessed using the SQL queries, programs etc. which were described in the DID. This step is the actual production process.
4. Post-processing: Result documents created with ITP can be post-processed to for example print the documents, produce PDF documents, e-mail the result documents, etc.

## 2.2 Database abstraction

ITP uses its own database descriptions for database access. In ITP language, these are called DID: **D**atabase **I**nterface **D**efinition. These database descriptions can contain multiple definitions of sets of related data (e.g. a database record) and the information required to retrieve this data. Relations between these definitions can be defined as well.

Through these definitions ITP models can access data without any knowledge of where the data resides or how to access it.

The DID is in fact an abstract view on a, possibly imaginary, database that maps to database files, external data retrieval programs or SQL queries. This mapping can be one-on-one, meaning that the abstract view closely mimics the original database, but it can also be completely different, offering a much more abstract (high-level) view of the database, which simplifies the development of model documents, especially when dealing with highly normalised databases.

By mapping the abstract view on the database to external programs, ITP enables the integration of self-written programs into the data retrieval, and therefore document production, logically. It furthermore allows developers to fine-tune the data retrieval process for special purposes, such as highly controlled data retrieval, mixed environment/DBMS data retrieval or high-performance data retrieval.

The phase of developing and implementing this abstract view is separated from the creation and execution of models, the document development and usage phase, enabling organisations to limit the data that are accessible from documents. This abstract view can also be modular, so that the DID or parts of the DID can be reused by other collections of documents.

The program that is used for this part of the ITP development process is called the ITP SDK.

## 2.3 Model documents, models and result documents

ITP works with three kinds of files in the text/data merge process:

- Model documents: These are word processor documents that can contain all kinds of text with all layout facilities offered by the word processor, mixed with ITP instructions. The ITP instructions enable data retrieval using external retrieval programs or logic, manipulation of data and merging those data into the document text.  
A model document is the equivalent of a program source.
- ITP Models: These are binary files generated from model documents, containing an optimised form of the model document, including all text, layout and ITP instructions. Model documents are compiled to ITP models. The model is the equivalent of a program.
- Result documents: These are normal word processor documents that contain the final document text, merged with -possibly manipulated- data. Executing ITP models produces these documents.

## 2.4 The ITP document development process

Developing a document in ITP means writing model documents (in your favourite word processor), compiling them to ITP models and testing them.

ITP program logic is written in the model document using the ITP instruction language, a powerful yet easy to use language, which has two very special properties, concerning input/output:

- Input is always done through a highly abstract and structured interface to the database or to external programs. The actual database access is not programmed in the model itself. Furthermore some user interaction can take place for additional input. All other input for the result document is already present in the model document itself! Other means of input are not (and do not have to be) available.
- Output is the result document itself, which means that all output produced is text and layout for the result document(s). Both may of course depend on database content. No special output facilities are needed.

The ITP instruction language furthermore offers all the usual programming facilities: functions, if-then clauses, while-loops, variables, arrays etc. It also offers powerful calculation, data and text manipulation and conversion functions. A model can be designed to produce one or more result documents.

The ITP instruction language is in fact a kind of programming language on word processor document level. Contrary to a normal programming language it is very easy to use (even by trained end-users), because of the special ways in which is dealt with input and output and because of the focus on text and layout in the word processor's native environment.

ITP supports model documents in several word processor formats: Word 6/95/97/2000/XP/2003, OpenOffice.org 1.1.1 and StarOffice 7.0, WordPerfect 6.1 and up and Ami Pro 4.0. (Lotus WordPro is supported through the Ami Pro document format.) Support is also available for HTML documents and ASCII documents.

## 2.5 Producing result documents

Executing the corresponding ITP models generates result documents; this is called the RUNMDL process. This process can be initiated from the ITP user interface or it can be integrated in an application, using ITP's APIs.

While running the model users can be asked to make (database) selections or to provide additional input. This is the case with interactive models. Models can also be designed to run without any intervention by the user.

During one run of the model, one or more result documents can be produced.

Performance of ITP document production is very good, determined mostly by how fast data can be retrieved.

The result document is in the same word processor format as the original model document.

## 2.6 Printing ITP output

ITP produces standard word processor documents. These documents are printed using the word processor itself as the print engine, thus allowing the total layout flexibility of modern word processors for your automated output production.

The word processor software uses the operating system's printing architecture to print documents. ITP can therefore be used with all printer-drivers that are available for the operating system. It can - for example- produce Postscript, PCL5, AFP and PDF output, or any other format generated by a Windows printer driver.

This open architecture also enables easy integration of ITP with -for example- fax solutions, web publishing solutions, archiving software and print servers.

## 2.7 The ITP product family

ITP is a product family of document production solutions that consists of the following products:

- ITP/Workstation for running ITP models for desktop based solutions.
- ITP/MDK for developing ITP models for desktop based solutions as well as server based solutions. The ITP/MDK includes the Model Development Kit. An extension to completely integrate the document development environment with Microsoft Word 97/2000/XP.
- ITP/MDK Repository: A model development environment combined with a document and ITP Model management system.
- ITP/Server to control document production batch processes, such as running ITP models, printing documents, converting documents, routing documents etc. Models for ITP/Server are developed with ITP/MDK.
- ITP/SDK AS/400 to describe database access for AS/400 databases.
- ITP/SDK MultiPlatform to describe databases access for local access programs, ODBC, Notes/Domino, Oracle, XML data files and mainframe access programs.

Both ITP/Workstation and ITP/Server support access to multiple database types:

- Lotus Notes Connection
- AS/400 Connection
- Oracle Connection
- ODBC Connection
- XML File Connection
- XML WebServices Connection
- Mainframe Connection

Data from multiple database sources can be used in a single model.

More information about these products can be found on our web site <http://www.aia-itp.com/>.

### 3. The ITP Instruction language

The ITP instruction language can roughly be divided into three parts:

1. instructions for the creation of text
2. instructions for data retrieval
3. instructions for data manipulation

To distinguish between text that has to appear in the result document and ITP instructions, the hash symbol # is used as a “tumble switch”; each time a hash symbol is encountered, ITP switches from “instruction-mode” to “text-mode” or vice versa. A model document will always start in text-mode, so before the first instruction a hash symbol has to be placed.

A very simple model document could look like this:

```
#
BEGIN

#
This is a very simple model document.
#

END
#
```

The BEGIN and END instruction indicate the beginning and end of the ITP instruction sequence.

This would result in a document containing the following text:

```
This is a very simple model document.
```

Since no data is being merged, this is of course not a very useful document.

If data has to be merged into the text, this can be done using the @-construct:

```
. . .
#
Dear Ms. or Mr. @(Cust.Surname),

In reference to you writing...
#
. . .
```

## ITP and ODBC

In the above example the field *Surname* from the record *Cust* will be merged into the document. But where does this record come from?

Data retrieval is achieved through the use of Entries, abstractions of the database which are defined in the DID. In the following example the entry *Customer* will retrieve data from the database. The fields that are retrieved are grouped in a 'record' that will be referred to as *Cust* and which is available between the ITP instructions DO and OD. The fields that are contained in the record have been defined in the entry definition in the DID. Since the method to actually retrieve the data has also been defined in the DID, no more specifications are needed in the model document.

```
...
WITH Cust IN EXP.Customer DO
#
Dear Ms. or Mr. @(Cust.Surname),

In reference to your writing...
#
OD
...
```

The WITH construct will retrieve one record from the database. If multiple records should be retrieved you can use the FORALL construct. The DO ... OD part of the FORALL construct will be executed for each record that has been retrieved.

Again, the method to retrieve the data has been defined in the DID.

```
...
WITH Cust IN EXP.Customer DO
#
Dear Ms. or Mr. @(Cust.Surname),

In reference to your writing....

A list of the ordered articles follows:
#
  FORALL Art IN Cust.Ordered_articles DO
#
  @(Art.Number_of_articles) @(Art.Article_description)
#
  OD (* FORALL Art IN Cust.Ordered_articles *)

OD (* WITH Cust IN EXP.Customer *)
...
```

The relation between *Customer* and *Ordered\_articles* has been defined in the DID. So, the developer of model documents only need to know that these relations exist, not how they are implemented.

## ITP and ODBC

To clarify ITP coding in the model document, comments can be added. Comment starts with (\*) and end with \*). Most word processors can also aid in the creation of easily readable documents by using different styles, colours, etc.

Of course the contents of a document should differ based on -for example- whether the customer is male or female.

```
...
Dear #
  IF Cust.Sex = "M" THEN # Mr. #
  ELIF Cust.Sex = "F" THEN # Ms. #
  ELSE # Ms. or Mr. #
  FI
# @(Cust.Surname),
```

In reference to your writing...

Although this gets the job done, it is not quite obvious what text will be produced in the result document. Introducing variables will make this somewhat easier.

```
...
TEXT ms_mr

IF Cust.Sex = "M" THEN
  ASSIGN ms_mr := "Mr."
ELIF Cust.Sex = "F" THEN
  ASSIGN ms_mr := "Ms."
ELSE
  ASSIGN ms_mr := "Ms. or Mr."
FI
#
Dear @(ms_mr) @(Cust.Surname),
```

In reference to your writing...

In this example a variable of type TEXT is declared. Then a value is assigned to this variable based on the contents of the field. Finally the contents of the variable are merged into the text using the @-construct. As a result, by separating the logic as much as possible from the text, the structure of the result text can be more *wysiwig*.

More than 60 build-in functions are available to format, convert or otherwise manipulate both numerical and text values.

```
...  
#  
@(Cust.Name)  
@(Cust.Street) @(Cust.Housenumber)  
@(uppercases(Cust.City))  
  
Nijmegen, @(date(today))  
  
...  
#  
...
```

Might result in:

```
Aia Software b.v.  
Kerkenbos 10-129  
NIJMEGEN  
  
Nijmegen, 12 April 2003  
  
...
```

In the above example three built-in functions are used: *uppercases*, *date* and *today*.

*Uppercases* will convert a text into the same text, but -as the name suggests- all into uppercases.

*Today* has no parameters and will result in a numerical representation of the current date. In the above example this is 20030412.

The function *date* will print a date with the name of the month “in words”. Of course the result of the date function depends on the language being used, something that can be changed while producing the text. This is especially useful for internationally operating companies.

It would go beyond the scope of this short introduction to show examples of all the other facilities the ITP instruction language offers.

## 4. ITP and ODBC

One single ITP model can access multiple ODBC databases.

### 4.1 Database access

ITP database access is defined by entries with fields and relationships between these entries. Refer to chapter 3 for more information. The document developer never sees any details about the actual definition of these entries. Entries are defined in the DID.

These entry definitions are usually generated with the ITP SDK Wizards. The generated definitions can be manually adapted or extended. The *DID Developer* manual, *ITP/SDK MultiPlatform Program* book contains a step-by-step explanation of these Wizards.

#### 4.1.1 Entries

##### Selection statements

Entries provide access to selected columns in ODBC tables. ODBC SQL statements are used to select the columns that should be retrieved. This SQL statement usually takes the following form:

```
FROM CUSTOMER
```

The SELECT part of the statement will be dynamically added by ITP.

ITP entries can return one single result (one row of columns) or multiple results. ITP entries can be parameterised. These parameters are used to limit the subset of the results that are retrieved. Thus the SQL statement can be extended with parameters that are passed to the entry when the entry is used:

```
FROM CUSTOMER WHERE CUSTOMERNUMBER=:1
```

In this example, **:1** denotes the first parameter to be passed to the entry. Entry parameters are defined with the entry.

##### Fields

An entry specifies a list of columns to retrieve from the database. The DID developer decides which fields should be retrieved. The fields are defined with their ODBC column name or with a special formula, e.g. to convert a numerical value to text using a mask.

##### Example

The example below is a complete entry definition that is stored in a DID document. This code is usually generated by Wizards and manually adapted or extended. It would go beyond the scope of this short introduction to explain all details of this example.

```

DEFINE_ENTRY
  NAME
    Customer
  MODEL_DOC_STATEMENT
    WITH

  DATA_RETRIEVAL
    "from CUSTOMER where CUSTOMERNUMBER=:1"
  KEY_RETRIEVAL
    "from CUSTOMER where CUSTOMERNUMBER>=:1"

  DEFINE_PARAMETERS
    Customernumber NUMERICAL (13 0)
  END_DEFINE_PARAMETERS

  DEFINE_FIELDS
    Customernumber NUMERICAL (13 0)  DATABASE_FIELD "CUSTOMERNUMBER"
    Surname         C_CHAR (31)      DATABASE_FIELD "SURNAME"
    Firstname       C_CHAR (31)      DATABASE_FIELD "FIRSTNAME"
    Initials        C_CHAR (11)      DATABASE_FIELD "INITIALS"
    Dateofbirth     NUMERICAL (11 0) DATABASE_FIELD "(
      {fn year(BIRTHDATE)} * 10000 +{fn month(BIRTHDATE)}
      * 100 + {fn dayofmonth(BIRTHDATE)} )"
  END_DEFINE_FIELDS
END_DEFINE_ENTRY (* Customer *)

```

### Stored procedures

Entries can also be based on ODBC Stored Procedures. The input parameters of the stored procedure become the formal parameters of the ITP entry and the output parameters or columns of the stored procedure become the fields of the ITP entry. Stored procedures that use input/output parameters cannot be used with ITP. The stored procedures may return multiple sets of output parameters, as long as the sets are equally formatted. In this case you should define the entry as a FORALL entry.

The ITP/SDK MultiPlatform provides a wizard to support the use of ODBC stored procedures.

## 4.2 Integration

ITP offers an extensive set of API's to control the ITP environment, to compile model documents and to run the merge process. The latter is the central ITP API, called RUNMDL. You must pass a number of parameters to this API. The most important are:

- name and path of the ITP Model
- name and path of the output document

You can also supply some additional control parameters. With these parameters you can control whether the document should be automatically printed or opened in the word processor. You can also supply parameters that can be used within the ITP Model. This means that you can supply the key information needed for the contents of the document, for example an invoice-number or customer id. The ITP Model will then, based on these parameters, retrieve all necessary information.

For integration with applications we provide an ActiveX Control. Alternatively you can also call functions in a dynamic link library (DLL) or call the ITP programs directly with command-line parameters.

**ITP** is developed by 

For more information please contact us.

**Telephone:** +31 24 371 02 30  
**Fax:** +31 24 371 02 31  
**WWW:** <http://www.aia-itp.com>  
**Email:** [itp@aia-itp.com](mailto:itp@aia-itp.com)  
**Postal Address:** P.O. Box 38025  
6503 AA Nijmegen  
The Netherlands  
**Visiting  
Address:** Kerkenbos 10-129  
6546 BJ Nijmegen  
The Netherlands