Rijksuniversiteit Groningen **Faculteit der Wiskunde en Natuurwetenschappen**

Vakgroep Informatica

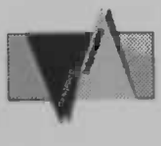
The INTELLIGEN™ 3-tier system

Subang Jaya, September 1996

Author:
Redmer Schukken std. nr. 0614912.
17b Jalan SS 15/5a
47500 Petaling Jaya
Selangor, Malaysia

Under supervision of:
Prof. Dr. Ir. L.J.M. Nieuwenhuis
S. Chong
J. Vong

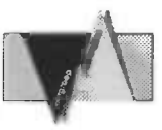
For:



RuG

The INTELLIGEN™ 3-tier system

By Redmer Schukken.
For:



Overview

This paper describes research on a 3-tiered network concept, which separates a network into 3 tiers, a back end database server tier, an application server tier and a front-end terminal tier, thus making it a combination of a Central Host and a Client/Server concept.

Conclusions:

- the implementation was successful.
- the performance does not increase with one Application Host. But if more than one Application host is used, performance is better than in the case of a Central Host architecture.

For Test Results see: Section 7 on page nr.54.

Preface

When I first came to Malaysia I did not know what to expect, but the same thing can be said about I&J, because they were expecting a Mat Saleh, and yet a Dutch guy appeared who looked like an Indonesian.

As I started doing some Clipper work and the big Agro job, I talked to Jimmy and Santo about what would be best for my final project. Apparently Jimmy had the 3 tiered idea already before I came, that is why the Advantage was already installed.

So after I finished the Agro job, we concluded that the 3 tiered system would probably be a good challenge for me, especially since I wanted to graduate in networks.

I had a great time at I&J and I would like to thank everybody at I&J and especially Jimmy, Ean and Santo for supporting me throughout my internship and of course for showing me all the local stuff one needs to see and taste in Malaysia. I think I learned more in these six months than I could have ever learned in Holland.

Furthermore I would like to thank my parents for supporting me and Rozemarijn for always having an ear for my problems.

Table of contents

1 Introduction

This thesis is written during my internship at I&J software. In this section we will give a brief introduction to the company I&J and their product Intelligen. We will briefly state the characteristics of Intelligen and its limitations, which leads to the final problem description.

1.1 Company description

I&J is a native Malaysian software company located in Kuala Lumpur that develops customized turnkey systems for medium sized businesses, which require business critical systems. The systems they develop are Automatic Transaction Processing systems (ATP). To do this they have developed a Rapid Application Development (RAD)-tool, called Intelligen, with which they can develop running systems in less than 3 months (depending on the system).
They have currently about 30 sites running with Intelligen.

1.2 Characteristics of Intelligen

The Intelligen engine is a script driven interpreter, which I&J calls a RAD-tool (Rapid Application Development). With Intelligen we can develop customized software by just formulating parameters, which are in fact expressions. Intelligen doesn't use any compiling nor linking, and is therefore very useful for remote support, since no new object code has to be uploaded to the customer site. Especially in a city like Kuala Lumpur with all its traffic jams. (See Concept of Intelligen on page nr. 9)

Intelligen currently runs on System Manager, a centralized host operating system (see Centralized Host Architecture on page nr.13). This operating system allows for very remote support. We can just login to the host via a modem. Once we are in, we can "possess" other users screens and "guide" them through any problems they might be having. We can also make some adjustments to the system by just altering the parameter files. Since no compiling nor linking is required, we don't have to upload the new object code, since there is none.

System manager runs on a standard PC, which makes it cheap, but which limits it also in computing power. With the current processors available in the market, the system becomes very slow if there are more than 8 concurrent users. This may be enough for certain departments of a company, but is certainly not enough for a whole company.

This leads us to the following objectives and approach:

1.2.1 Objectives

Port the Intelligen engine to a new architecture so that:

- More concurrent users can access the Intelligen
- The same remote support can be given as in the current system

1.2.2 Approach

- Give an overview of possible architectures to port the Intelligen to
- Give a description of how Intelligen works, and how it uses parameter files to build an application
- Design and implement Intelligen on the new architecture
- Test the new system and compare it to the old one

These objectives and this approach leads us to the following problem description:

1.2.3 Problem description

How can the Intelligen 3-tier Posting Engine be designed and does it improve the Central Host concept in larger networks?

1.2.4 Structure of the thesis

We will first describe the concept of Intelligen in section 2, followed by a review of different architectures to port the Intelligen to in section3.

After that we will go into detailed description of Intelligen in section 4 followed by section 5, Design and section 6 Implementation.

Finally we will show the test results in section 7and give our conclusion in section 8.

This thesis comes with two other documents: ABP Source code, which contains all the source codes, and the ABP Users Manual.

2 Concept of Intelligen

In this section we will describe the concepts of the Intelligen engine. First we will describe the concept of zero-programming, and its parameter-driven basis. Further in this section we will describe how workflow is captured in the Intelligen engine.

2.1 Approach to ATP (Automated Transactions Processing)

2.1.1 Zero Programming

Zero Programming is meant to reduce code writing to formulating expressions in Parameter files. This way there is no code which needs to be compiled, nor are there any objects that have to be linked.

It means that the implementation of Customized software can be done much faster, since there is far less need for debugging, plus the implementation becomes much more transparent to other developers.

With conventional coding, the readability of the code depends severely on the code writer. A good code writer will add a lot of annotations to make the code more readable. A bad code writer on the other hand might not enter any annotations at all.

Since writing programs is reduced to writing expressions in parameters, the implementations are very transparent, since they cannot be more complex than an expression.

2.1.2 Parameter driven concept

You can look at parameter files as resource files in Unix systems, but more sophisticated. The parameter files are loaded into memory at startup. The engine in this case functions as an interpreter, which interprets the expressions when they are read into memory.

To make alterations the developer goes "into" the engine with a password. He can then make all the necessary alterations in the parameter files, and on exit, which gets him back to the menu, all the alterations are updated into memory. So in fact there is even no need to restart the program.

This is very useful for customer support. When, for instance, a customer wants a field altered, the developer can just dial in, using a modem, and have the customer point out what he exactly wants altered. He then enters the application with a password and makes the alterations. When he exits, the changes are already done. So he does not have to recompile and relink the application and upload the whole object code.

2.2 Workflow capture and analysis

Intelligen provides a uniform way to capture workflow and to analyze data. We will briefly go through the basic components.

We start the capture of a workflow with the most basic object, which is a business form. Business forms are documents used to take orders from customers, or to issue invoices, etc. Within Intelligen business forms are called **Documents**. Documents contain dynamic data, since they are changed very often.

Within a document we want to validate data against other data. For instance on an order form we want to check the customer that places the order against a customer list, so that we only have to issue the customer number, and get the address etc. automatically. And we want to check the products he orders against a product price list. Customer lists and product price lists are called **Profiles**. Profiles contain semi static data, since they are changed, but not very often. A customer may change his address, but not on a daily basis.

Finally within the profiles we want to validate data against static data. For instance if we want to define the gender of a customer, we want it to be checked against a gender table, which contains either M or F. To make sure the user type "M" instead of maybe "m", we use a Table to check this Or we want to check the state he is living in against a state table. Gender tables and state tables are called **Tables**. Tables contain static data, since they are (almost) never changed.

Figure 1 shows the relation between documents, profiles and tables.

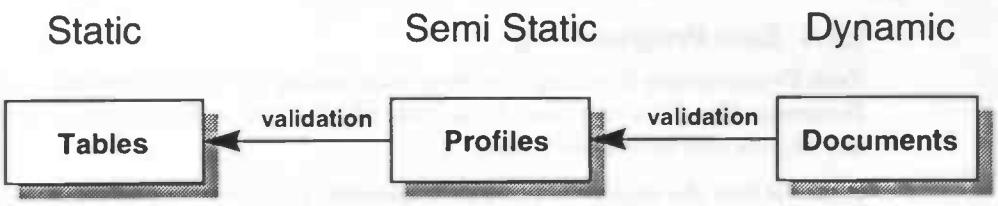


Figure 1: Validation process

We define tables, profiles and documents as follows.

2.2.1 Documents

Documents are databases that contain dynamic data, which means that they are generated every day. Documents correspond to business forms in companies.

Typical documents are Delivery Orders, Invoices, Service Orders, etc.

2.2.2 Profiles

Profiles are databases that contain data that is semi-static, which means that they may have to be changed from time to time, but not all the time.

Typical Profiles are Customer Lists, Supplier Lists, Product Price List, etc.

2.2.3 Tables

Tables are databases that contain the most elementary static data elements. Elements that cannot be divided into smaller elements. Tables normally contain very little data and are basically used for validation. The data in tables is very static.

Typical tables are items like Country Code, Area Code, Gender, etc.

Now that we are able to capture the basic workflow, we also want to analyze the captured data and view it. There are two ways to view analyzed data: as printed **reports** or as onscreen **enquiries**.

For instance at the end of the month we want to print a monthly sales report, which is basically a summation of all the invoices issued. Or if we want to check what a customer has ordered, we can use an enquiry to view his orders.

But before we can generate reports and enquiries, we have to combine the data captured in documents in a way that we can actually generate the different reports and enquiries. For this we use a new data entity which we call **Masters** and combining data from documents into a master is called **Posting**.

Figure 2 shows how data from documents is posted into masters, and viewed in reports and enquiries.

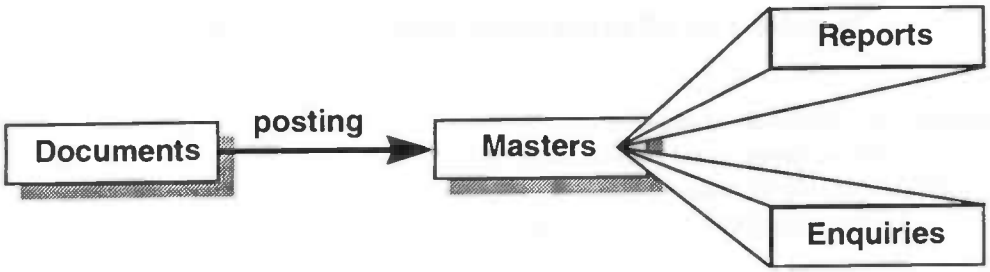


Figure 2: Relation between Documents, Masters, Reports and Enquiries

We define reports, enquiries, masters and posting as follows:

2.2.4 Reports

Reports are lists or forms created from data captured in documents or posted into masters, which can be specified by range. Reports can either be viewed on screen or printed.

Typical reports are Invoices, Delivery orders (which are forms), Sales Analysis Report, Physical Stock Report (which are Reports/Listings).

2.2.5 Enquiries

Enquiries are onscreen, on-line views of data, which can be specified per data entry.

Typical Enquiries are Stock Update Enquiries, Debtor Cards, Product Enquiries etc.

2.2.6 Masters

Masters are data entities in which data is combined from documents in a way that they can be viewed as reports and/or enquiries.

Typical Masters are Physical Stock Master, Sales Analysis Master, General Ledger, Debtors Ledger, etc.

2.2.7 Posting

Posting is the combining of data from documents into masters in a way that they can be viewed as reports and/or enquiries.

Posting can be done on-line, which means that the data is posted directly after the user has keyed in a document, or batch, which means that the data is posted by range.

2.3 Different Engines within Intelligen

The Intelligen engine consists of different sub-engines corresponding to the different components of the engine, which are all started from the menu-engine.

Figure 3 shows these sub-engines. On top we see the menu-engine to generate menu's. Underneath we see the database-engine, which is used to generate tables, profiles, documents and masters and the report- and enquiry-engine are used to generate reports and enquiries. Further the validation-engine to generate validations, and the posting- and batch-posting-engine to generate posting parameters.

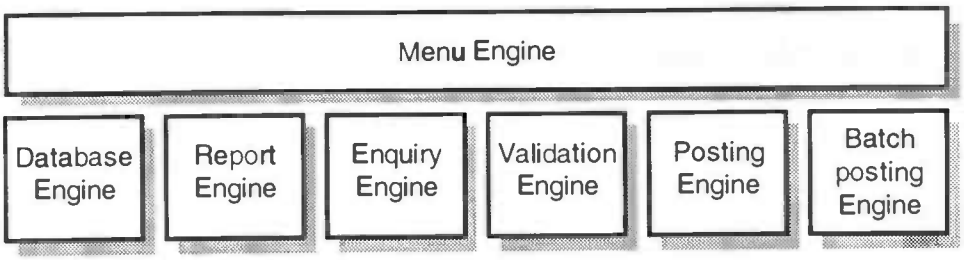


Figure 3: Different sub-engines of Intelligen

In section 4 we will go into further detail about the Intelligen engine.

3 Architectures

In this section we will give an overview of architectures mostly used today, distributed single user architecture, centralized host architecture and client/server architecture, plus a new architecture which we will call a 3-tiered architecture. At the end of this section we will choose one of these architectures to port the Intelligen engine to.

3.1 Introduction

Before we will describe different architectures, it is good to give a definition of architecture. Tanenbaum describes computer architecture as: the set data-types, operations and facilities on each level of a system[1].

3.2 Distributed Single-user Architecture

A distributed single-user architecture can be described as an architectural approach that designs an application so that all functional components of the application reside on a single computing platform dedicated to the use of only one person at a time with a limited form of simultaneously shared access to data across the platforms interconnected by the network[2].

Figure 4 shows a typical distributed single user architecture. On the left we see two computers with each a single uses. Both have limited access to shared resource, on the right.

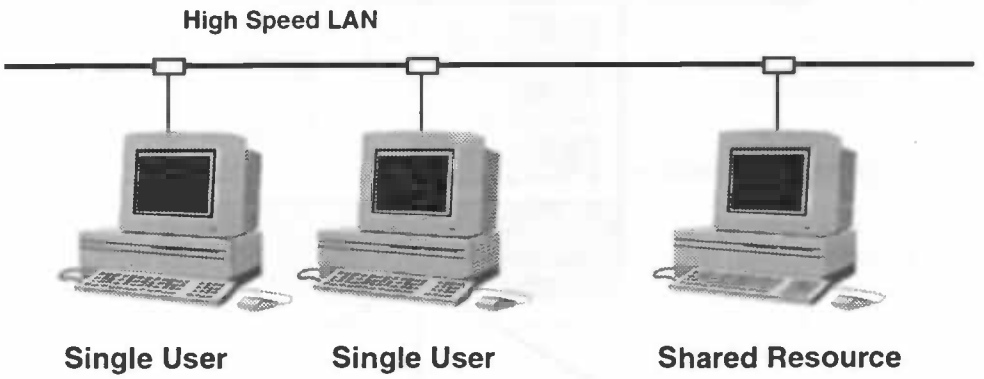


Figure 4: Distributed Single User Architecture - Hardware setup

This platform is not very widely used for transaction processing, not only because the network can get congested, since every disk access has to go through the network, but more because there is very little concurrency control in the shared resource.

3.3 Centralized Host Architecture

A Centralized Multi-user Architecture can be described as an architectural approach that designs an application so that all functional and data components of the application reside and execute upon one centralized computing platform[3].

Figure 5 shows a typical centralized host architecture: a single central computing platform with a number of (dumb)-terminals connected to it with a low speed network.

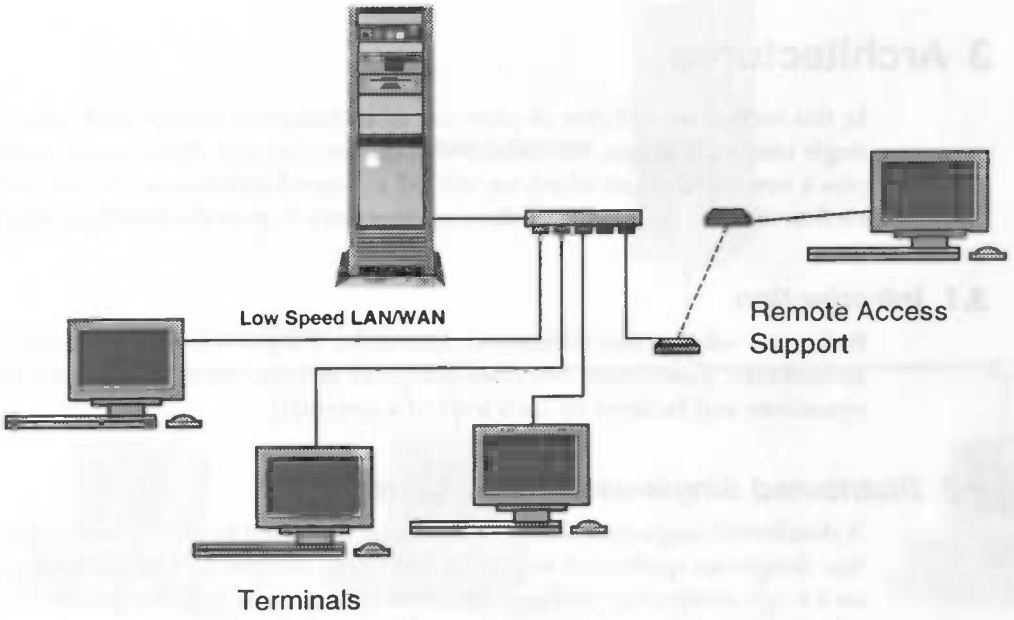


Figure 5: Centralized Host Architecture - Hardware setup

The reason that the network can be low speed is that the only data transmitted over the network is keyboard and screen updates (and optionally mouse and printer). Figure 6 shows the allocation of software functions in a central host architecture.

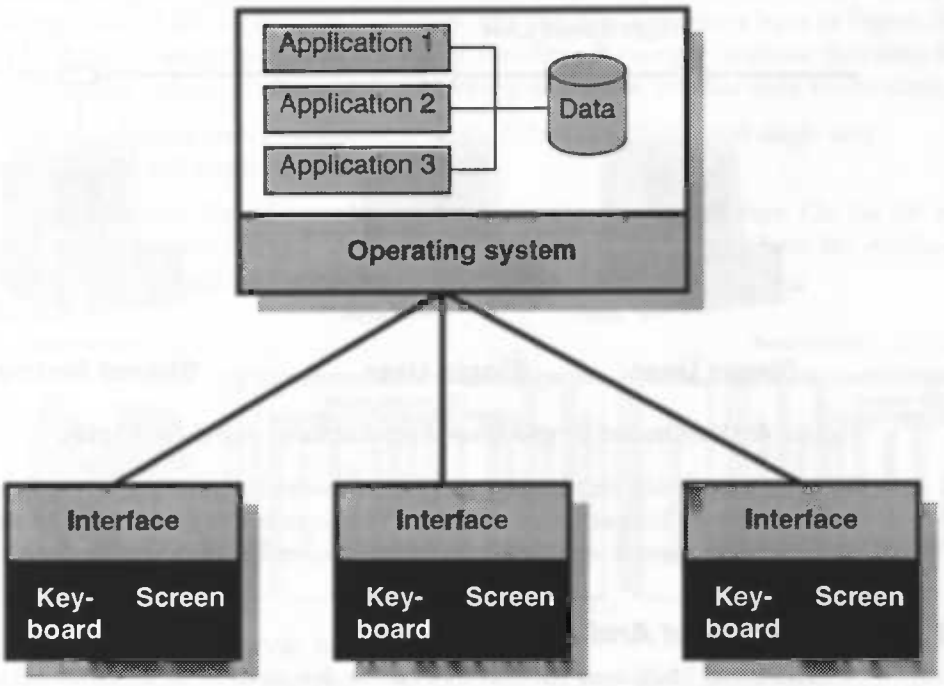


Figure 6: Centralized host architecture - Software components

Figure 7 shows the different layers of the centralized host platform. On the left we see a terminal and on the right the central computing platform, connected through a low speed network.

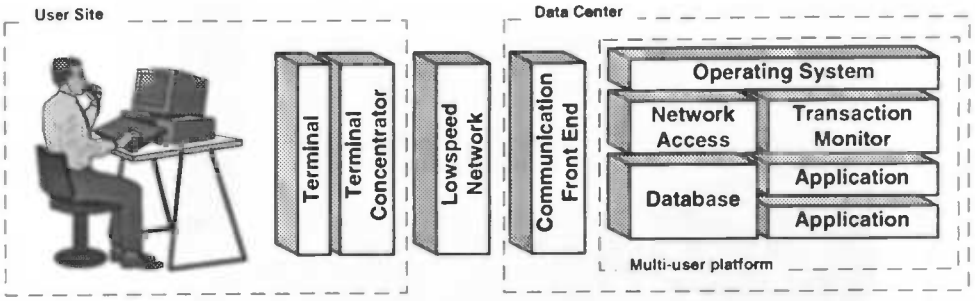


Figure 7: Layers of the centralized host architecture

The basic problem with a Centralized Host is that it limits the number of users. After 80% usage of the designed capacity, performance drops drastically as Figure 8 illustrates.

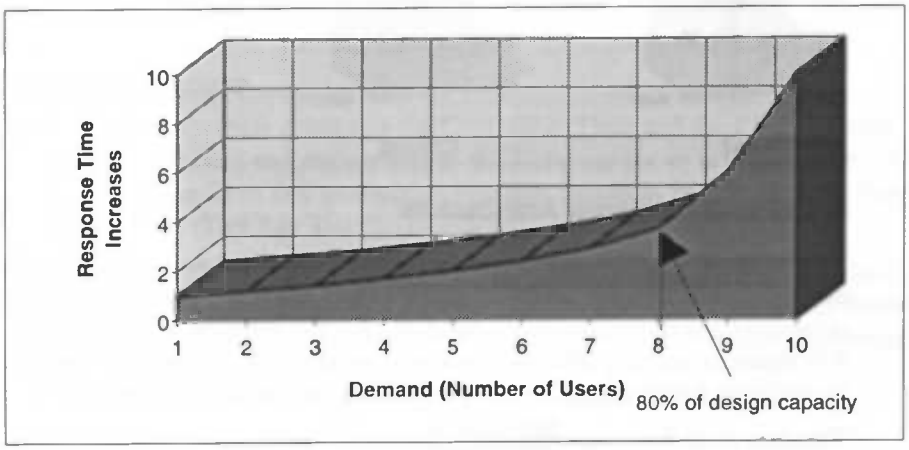


Figure 8: Centralized host architecture performance characteristics[4]

To increase the number of users, the complete host has to be upgraded which can be very costly, if at all possible. The current Operating System that Intelligen uses, a system called System Manager, limits the number of processors to 1, so there is an absolute limit to the number of users, which currently stands at about 8.

3.4 Client/Server Architecture

This approach designs an application so that the functional components of an application are partitioned in a manner that allows them to be spread across and executed on, multiple different computing platforms sharing access to one or more common repositories of data[5].

Figure 9 shows a typical client/server architecture. On top we see the database server which is connected to the clients through a high speed LAN.

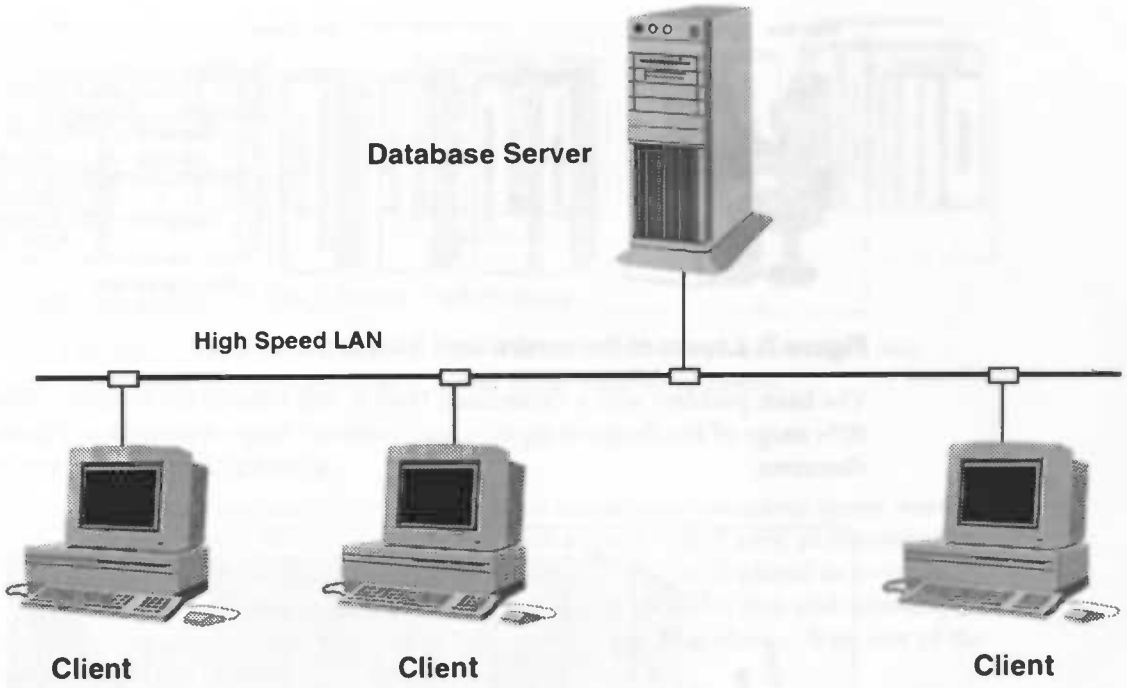


Figure 9: Client/Server Architecture

The actual application resides on the client. If the client wants access to the shared databases, which reside on the database server, it packs this request as a query, which is unpacked at the server (this conforms with the database access layer in Figure 10). The database server performs this query, handling all primary database functions like (re)building indexes, seeking and (un)locking, and sends the data back to the client.

This way there is better concurrency control than in a distributed single user architecture and less usage of the network.

Figure 10 shows the different layers of the client/server architecture. On the left we see the client platform and on the right the server platform. Both have the database access layer to pack and unpack queries.

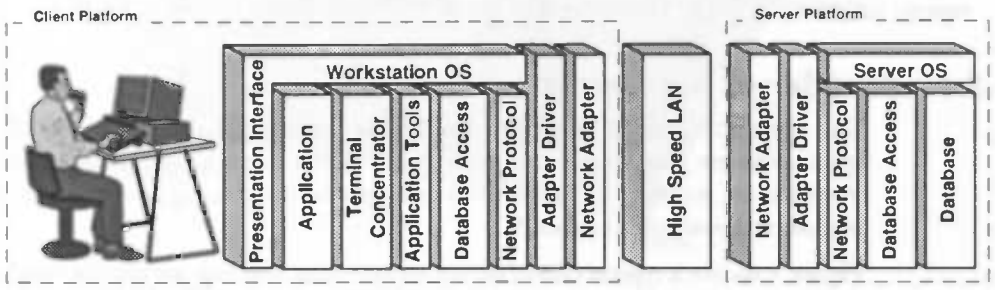


Figure 10: Client/Server Architecture

The advantage of the Client/Server architecture is that there is no real limit to the number of users. The Demand-Response time is linear, as illustrated in Figure 11. (Off course with increasing number of users, there will eventually be waiting time for data-locks, which will cause the response time to increase more than linear).

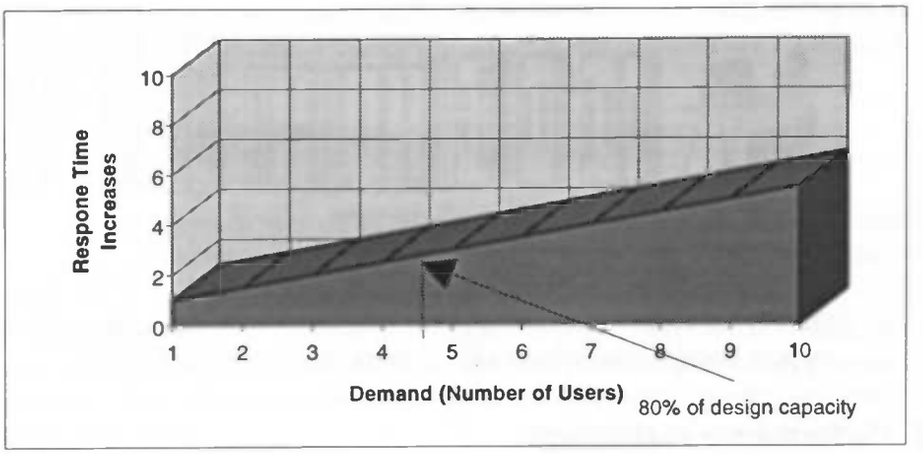


Figure 11: Client/Server architecture performance characteristics[6]

3.5 3-tiered Architecture

The 3-tiered architecture combines the Centralized Host and the Client/Server architecture. The back end connection to the Database server is Client/Server (Tier 1 to Tier 2), and the front end connection between terminals and Application Server is Centralized Host (Tier 2 to Tier 3).

Figure 12 shows a typical 3 tiered architecture. On top we see the backend database server (tier 1), connected through a high speed LAN with the frontend application servers (tier 2) and finally connected through a low speed WAN with the terminals (tier 3).

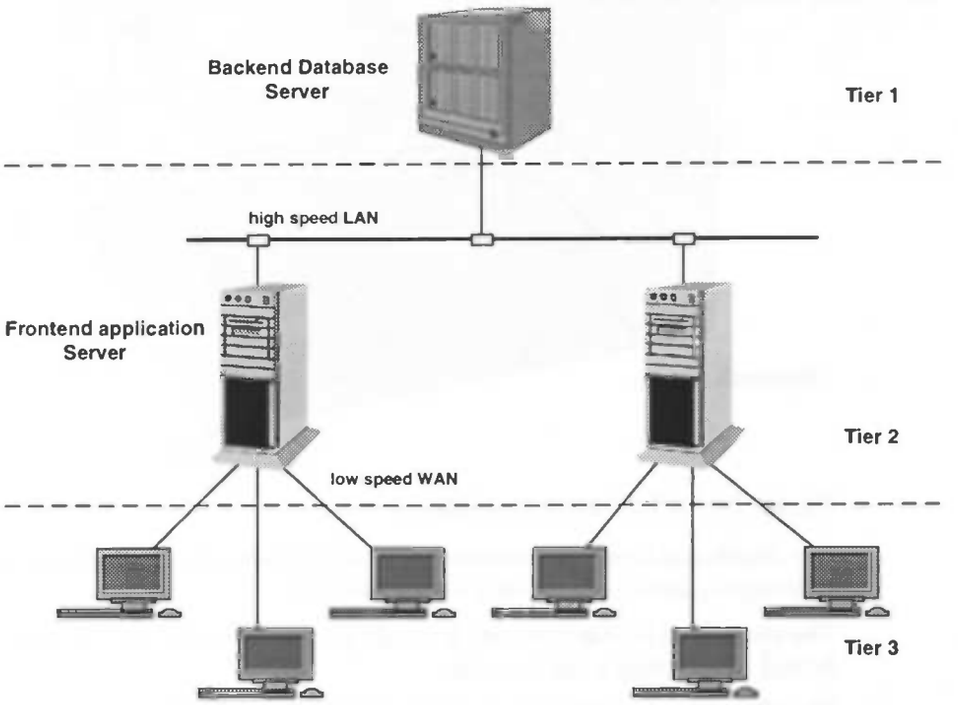


Figure 12: 3 Tiered Architecture

Figure 13 shows the different layers of the 3 tiered architecture. On the right we see the database server, in the middle the application server and on the left the user terminal.

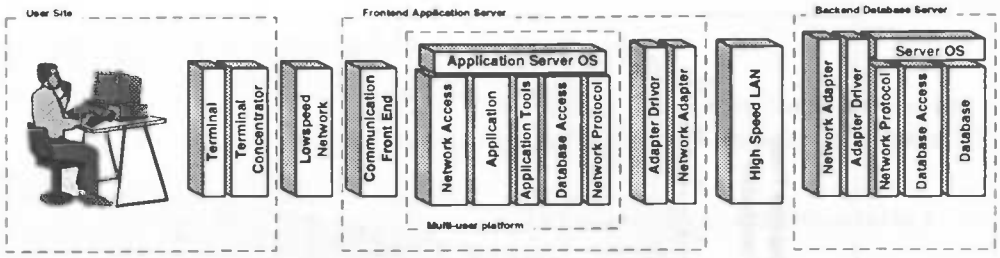


Figure 13: Layers of the 3 tiered Architecture

The 3 tiered architecture combines the benefits of a centralized host (direct user support through modem) and client/server (large number of users).

3.6 Performance restrictions

If we look at performance, there are three basic restrictions: processor speed, network speed and disk speed. We can illustrate this with a graph with 3 axes, as illustrated in Figure 14. The performance of each architecture can now be displayed as a vector: The components of the vector on the resource axes (processor, disk and network) represent the usage of that resource at full system usage. Therefore at least one of the components will be 100%.

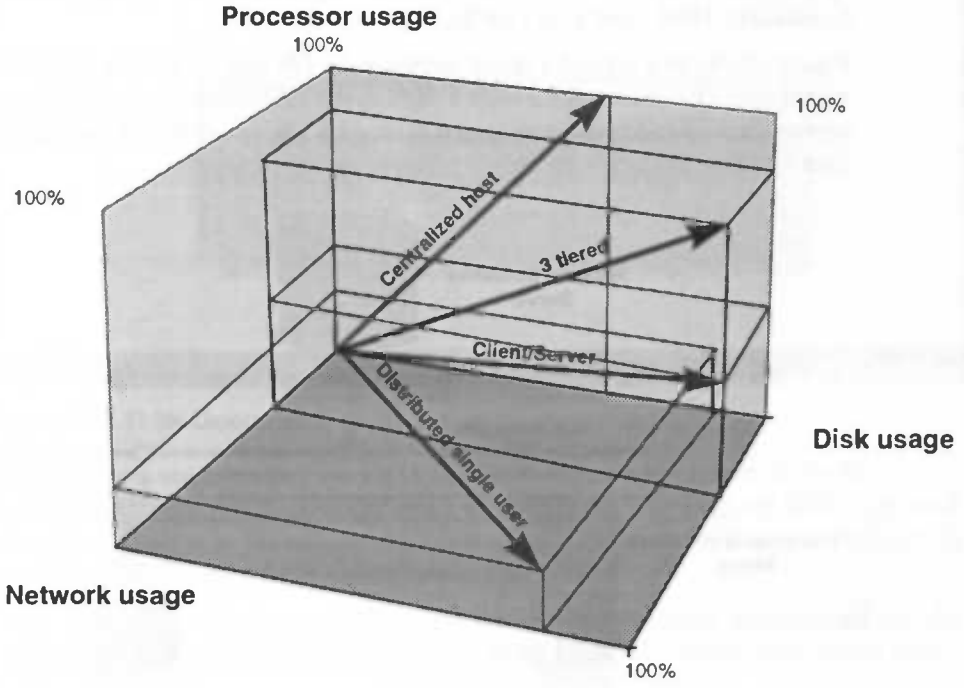


Figure 14: Performance restrictions

The distributed single user architecture is mainly network and disk bound, since all data request are directly handled via the network.

The centralized host architecture is mainly processor bound and not at all network bound, since it uses a star network.

The client/server architecture is mainly disk bound, since all the disk access is handled by the server, and less processor and network bound, since the number of processors increases with the number of clients and all data requests are "packed" into queries.

The 3 tiered architecture is both disk bound and processor bound, although this largely depends on the number of application servers. Also we have 2 networks: the

star network between tier 2 and 3 and the LAN between tier 1 and 2. In Figure 14 the network usage axis represents the high speed LAN that connects tier 1 with tier 2.

3.7 New platform for Intelligen

3.7.1 Choosing a new platform

The Intelligen currently runs on an operating system called System Manager, which is a Central Host platform. This is a multi-tasking multi-user DOS OS, which runs single processor PC's. This limits the number of concurrent users to about 8.

The first alternative one thinks of is of course a Client/Server platform, since this does not really limit the number of users. The only problem with a Client/Server platform is that it does not provide the option to dial in with a modem and give customer support.

With System Manager, dialing up to a (customer-)site and "possessing" other users' screens is possible. This way users can be guided through certain steps if they are stuck. With a Client server platform this option is not available.

In a 3-tiered architecture dial-up support and more concurrent users are available. That is why we have chosen the 3 tiered architecture as the new platform for Intelligen.

We will use System Manager as the operating system for the application server. The database server will be an Advantage Database Server running under Novell Netware.

3.7.2 Proposed implementation

Since I&J didn't want to release all the source codes for the Intelligen and since we only had half a year to complete this paper, we have decided to only implement the Automatic Batch Posting engine (ABP), which is the most critical part to port to the new platform.

4 Detailed description of Intelligen

In this section we will describe in more detail some of the parts of Intelligen that are relevant for the automated batch posting engine. First we will describe documents, followed by masters and posting.

Because Intelligen has its own definition of a Table, we will call a database table a database file.

4.1 Types of documents

Documents are used to capture data in Forms. There are two kinds of documents: THIN and FAT.

4.1.1 THIN document

Thin documents are the simplest form of a document. They basically consist of a Document Number, which is the key, and input fields that the user has to key in.

Figure 15 shows the input screen for a THIN document in Intelligen.

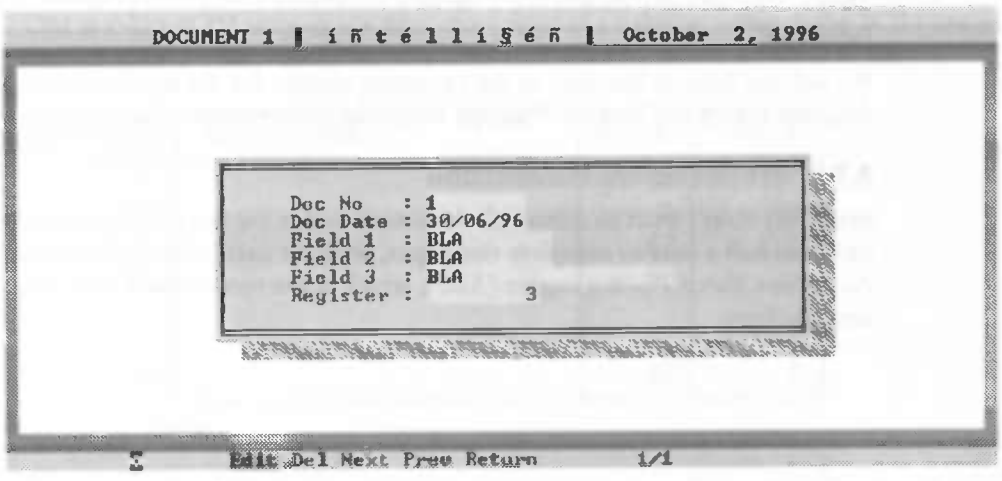


Figure 15: THIN Document

The minimum requirements for a THIN document are a key (most of the times DOC_NO), a date (DOC_DATE) and a Register field. The Register field is updated when the user keys in the information. It is taken from a central Register database file which handles requests from simultaneous users.

Every time a user asks for a new Register, the central Register is incremented and the new value is returned. Updating registers is done in INPUT mode only, not in EDIT mode.

The register field is used to determine which documents to post first and in what order they should be calculated. It is like taking a number rather than to stand in queue for a bank counter.

4.1.2 FAT document

A FAT document consists of a THIN and a FAT part. In the THIN part the information is stored which applies to the whole document (for example the customer you want to invoice). The FAT part is used to store information about one or more items.

FAT documents are for instance used for invoices which contain more than one item.

Figure 16 shows the input of Intelligen for a typical FAT document. Above the horizontal line are THIN fields and underneath are the FAT items.

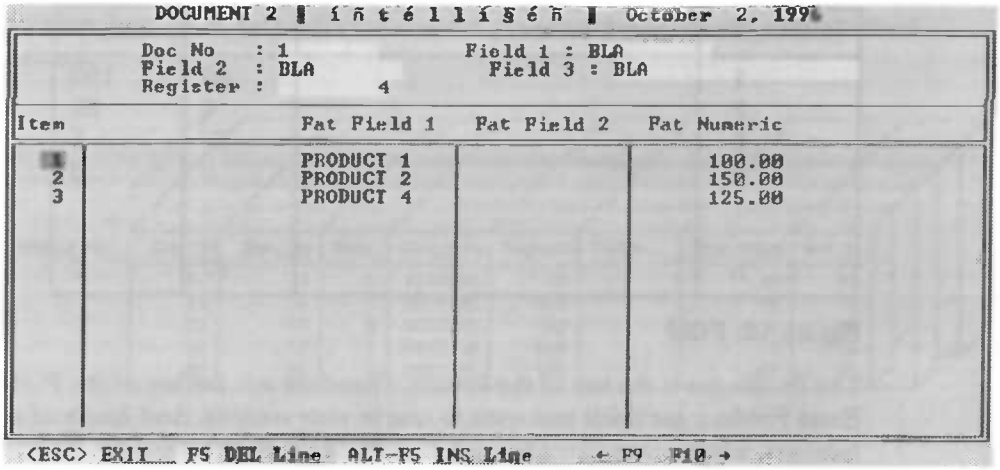


Figure 16: FAT document

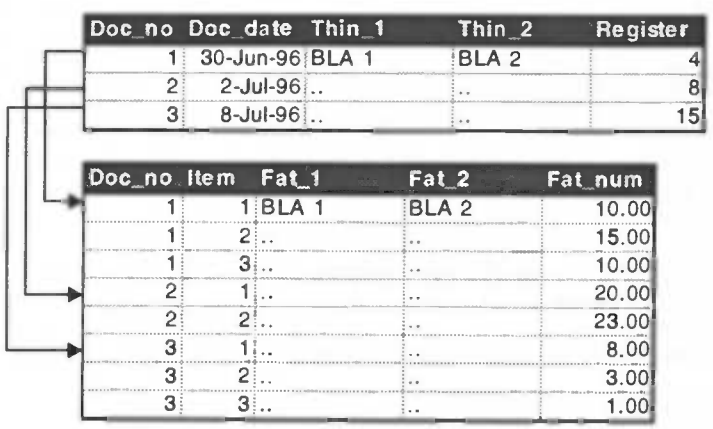
Physically a FAT document consists of 2 database files, a THIN database file and a FAT database file, containing respectively the THIN fields and the FAT fields.

The minimum requirements for the THIN part are the same as a normal THIN document (see THIN document on page nr. 20).

The minimum requirements for a FAT file are a DOC_NO field (or the key of the THIN part) and an ITEM field. The key should be DOC_NO+ITEM.

Finally there is a relation between the THIN document and the FAT document based on DOC_NO.

Figure 17 shows the concept of a FAT document. The upper table represents the THIN database file with the THIN entries. The lower database file represents the FAT database file with the FAT items. The arrows represent the relation between the two.



Doc_no	Doc_date	Thin_1	Thin_2	Register
1	30-Jun-96	BLA 1	BLA 2	4
2	2-Jul-96	..	..	8
3	8-Jul-96	..	..	15

Doc_no	Item	Fat_1	Fat_2	Fat_num
1	1	BLA 1	BLA 2	10.00
1	2	..	..	15.00
1	3	..	..	10.00
2	1	..	..	20.00
2	2	..	..	23.00
3	1	..	..	8.00
3	2	..	..	3.00
3	3	..	..	1.00

Figure 17: FAT document concept

4.2 Types of masters

Masters are used to keep track of the status of different profiles or documents. There are several kinds of Masters, which we will describe next.

4.2.1 POM (Profile On-line Master)

The POM keeps track of the status of an individual profile entry. For each profile entry the POM will have an entry, even if the balance is 0. Figure 18 shows a typical POM.

Profile_no	Extra Fields....	Bal_1	Bal_2
1		99	150
2		0	56
3		87	0
4		0	0
.		.	.
.		.	.
n		87	0

Figure 18: POM

The Profile_no is the key of the Profile. Therefore it is the key of the POM as well. Extra Fields... are fields you want to use in your analysis. And finally of course the balances, which are updated by each document.

The minimum requirements for a POM is a Profile Number field plus an index on this Profile Number.

4.2.2 SOM (Status On-line Master)

A SOM is almost the same as a POM. The only difference between a SOM and a POM is that a SOM only stores data when there is a balance, whereas the POM has an entry for each Profile, regardless of a balance. Figure 19 shows a typical SOM.

Status_no	Extra Fields....	Bal_1	Bal_2
4		99	150
9		0	56
11		87	0
12		67	55
.		.	.
.		.	.
n		87	0

Figure 19: SOM

The Status_no field is the key of the Profile or Document you want to keep track of. Therefore it is the key of the SOM as well. Extra Fields... are fields you want to use in your analysis. And finally of course the balances, which are updated by each document.

SOM's are mainly used to quickly check balances, for instance in an ATM machine a SOM can be used to check whether the customer is allowed to make a withdrawal.

The minimum requirements for a SOM is a Status Number field plus an index on this Status.

4.2.3 TSOM (Tracking Status On-line Master)

The TSOM (pronounced as TrackSOM) is the most commonly used master. It is basically a SOM, but it also stores the previous transactions that led to the current balance.

For instance in a banking system, a SOM will only show the account balance (when you check your balance at the ATM machine), whereas a TSOM will give the full bank account details with the individual transactions (like deposits, withdrawals, etc.) as shown in your bank receipt.

Figure 20 shows the concept of a typical TSOM. On the left we see the Status_no's. Per status there are transactions that led to the current balance, which is the last line for that Status_no. The scope of one Status_no can have a varying number of records,

because there could be a different number of transactions that led to the current balance.

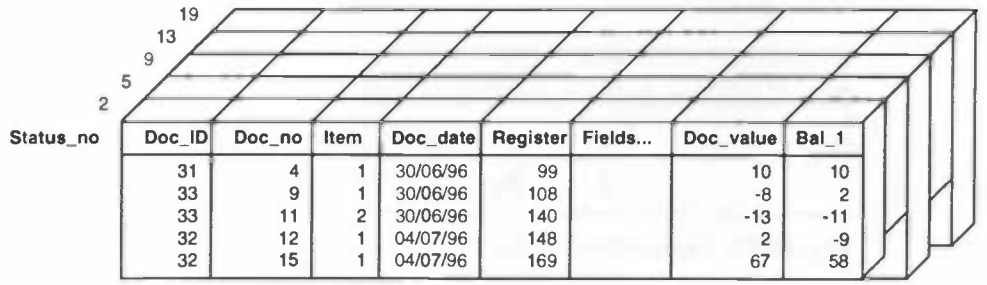


Figure 20: Concept of a TSOM

Doc_ID specifies the type of document (for instance 31 could be a delivery order document), Doc_no specifies the document number.

The transactions are indexed on Doc_Date+Register to calculate the new balance. But now the question of course is how this can be implemented physically. In order to do this some indexes have to be created and some fields added to the master. Figure 20 shows how a TSOM is physically implemented.

Doc_ID	Doc_no	Item	Doc_date	Register	Status_no	Fields...	Doc_value	Bal_1	Flag1
31	4	1	30-Jun-96	99	2		10	10	F
33	9	1	30-Jun-96	103	2		-8	2	F
33	9	2	30-Jun-96	103	2		-13	-11	F
32	12	1	4-Jul-96	128	2		2	-9	F
32	15	1	4-Jul-96	129	2		67	58	T
31	3	1	30-Jun-96	58	4		13	13	F
33	10	1	5-Jul-96	120	4		-4	9	F
33	10	2	5-Jul-96	120	4		-5	4	T
31	5	1	30-Jun-96	33	5		15	15	F
32	1	1	30-Jun-96	98	5		3	18	F
32	2	1	5-Jul-96	129	5		6	24	F
32	8	1	6-Jul-96	133	5		5	29	F
33	11	1	8-Jul-96	159	5		-9	20	T

Figure 21: TSOM physical form

In order to calculate the balance (BAL_1) a calculation index has to be created. This index is on STATUS_NO+DOC_DATE+REGISTER, where DOC_DATE+REGISTER is the time dependent factor. (The register is needed to separate documents keyed in on the same date).

Further a balance field has to be created, (BAL_1) in this case, a Status field (STATUS_NO), a document value field (DOC_VALUE) and a flag field (FLAG1).

The document value field is the value with which to update the balance field. The flag field is to flag the latest balance.

The balance is calculated as follows:

1. If the STATUS_NO of the previous record is not equal to the current record or it is the first record, then the previous_value=0, else previous_value=BAL_1 of the previous record.
2. Add Doc_value to the previous value and store it in the current record under BAL_1
3. Replace FLAG1 with .F.
4. Do steps 1 and 2 while Status_no is the same

5. If you have reached the last record with the same STATUS_NO set FLAG1 to .T.

as illustrated in Figure 22.

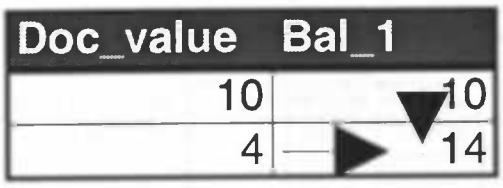


Figure 22: Calculation of the Balance field in a TSOM

4.2.3.1 TSOM with multiple balances

To make it even more complex, one TSOM can have multiple balances. For instance in an inventory master, it may be necessary to keep track of the balance per godown and per godown per product. This means that the balance of a godown and the balance of a product in that godown has to be known at any time..

Figure 23 shows the concept of a TSOM with multiple balances. We see the TSOM in 2 different forms, one for Status_1_no and one for Status_2_no, which can both be seen as a TSOM with a single balance.

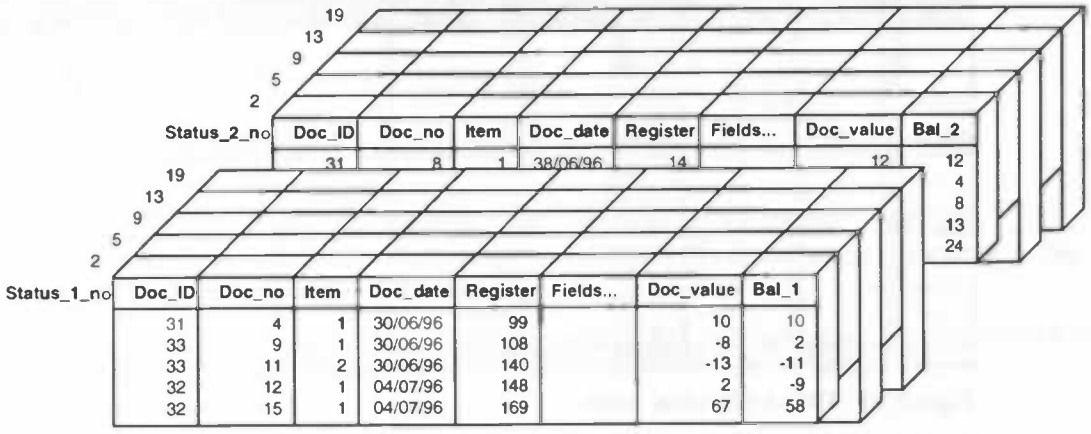


Figure 23: Concept of a TSOM with multiple balances

To implement this physically an extra balance field, to store the balance, an extra flag field, to flag the latest balance and an extra calculation index have to be created..

In the inventory example balance 1 by godown and balance 2 by godown and product have to be computed. Hence the first calculation index will now be:
GODOWN+DOC_DATE+REGISTER The second calculation index will be:
GODOWN+PRODUCT+DOC_DATE+REGISTER.
In both cases DOC_DATE+REGISTER are the time dependent factors.

First BAL_1, the first balance, is calculated by setting the index of the Master to the first calculation index. Then BAL_1 and FLAG1 are calculated in the same way as for a single balance.

Figure 24 shows the result of the first balance calculation.

Doc_ID	Doc_no	Item	Doc_date	Register	Godown	Product	Doc_value	Bal_1	Bal_2	Flag1	Flag2
31	4	1	30-Jun-96	99	GD 1	PD1	10	10		F	
31	8	1	30-Jun-96	120	GD 1	PD2	8	18		F	
33	9	1	30-Jun-96	120	GD 1	PD1	-2	16		F	
33	12	1	4-Jul-96	128	GD 1	PD2	-4	12		F	
31	15	1	4-Jul-96	129	GD 1	PD2	15	27		T	
31	3	1	30-Jun-96	58	GD 2	PD1	13	13		F	
31	10	1	5-Jul-96	120	GD 2	PD2	12	25		F	
33	16	1	5-Jul-96	120	GD 2	PD2	-5	20		T	
31	5	1	30-Jun-96	33	GD 3	PD1	15	15		F	
33	1	1	30-Jun-96	98	GD 3	PD1	-3	12		F	
31	2	1	5-Jul-96	129	GD 3	PD2	8	20		F	
33	8	1	6-Jul-96	133	GD 3	PD2	-5	15		F	
33	11	1	8-Jul-96	159	GD 3	PD1	-9	6		T	

Figure 24: TSOM set to first calculation index

The first highlighted column is theStatus field, the second the balance and the third the flag.

To calculate the second balance, the index is set to the second calculation index and BAL_2 and FLAG2 are calculated in the same way as a single balance.

Figure 25 shows the TSOM after the calculation of the second balance.

Doc_ID	Doc_no	Item	Doc_date	Register	Godown	Product	Doc_value	Bal_1	Bal_2	Flag1	Flag2
31	4	1	30-Jun-96	99	GD 1	PD1	10	10	10	F	F
33	9	1	30-Jun-96	120	GD 1	PD1	-2	16	8	F	T
31	8	1	30-Jun-96	120	GD 1	PD2	8	18	8	F	F
33	12	1	4-Jul-96	128	GD 1	PD2	-4	12	4	F	F
31	15	1	4-Jul-96	129	GD 1	PD2	15	27	19	T	T
31	3	1	30-Jun-96	58	GD 2	PD1	13	13	13	F	T
31	10	1	5-Jul-96	120	GD 2	PD2	12	25	12	F	F
33	16	1	5-Jul-96	120	GD 2	PD2	-5	20	7	T	T
31	5	1	30-Jun-96	33	GD 3	PD1	15	15	15	F	F
33	1	1	30-Jun-96	98	GD 3	PD1	-3	12	12	F	F
33	11	1	8-Jul-96	159	GD 3	PD1	-9	6	3	T	T
31	2	1	5-Jul-96	129	GD 3	PD2	8	20	8	F	F
33	8	1	6-Jul-96	133	GD 3	PD2	-5	15	3	F	T

Figure 25: Calculating the second balance

The first highlighted columns are the Status fields, the second the balance and the third the flag.

To view the different balances, the index is simply reset to the desired one.

4.2.4 FOM (Frequency On-line Master)

A FOM is used to store the cut-off points of a SOM or a TSOM at the end of each period.

For instance at the end of the month, the closing balances of each account can be stored into a FOM.

Figure 26 shows the concept of a FOM. On the left we see the closing dates. Per closing date there are entries for each status_no.

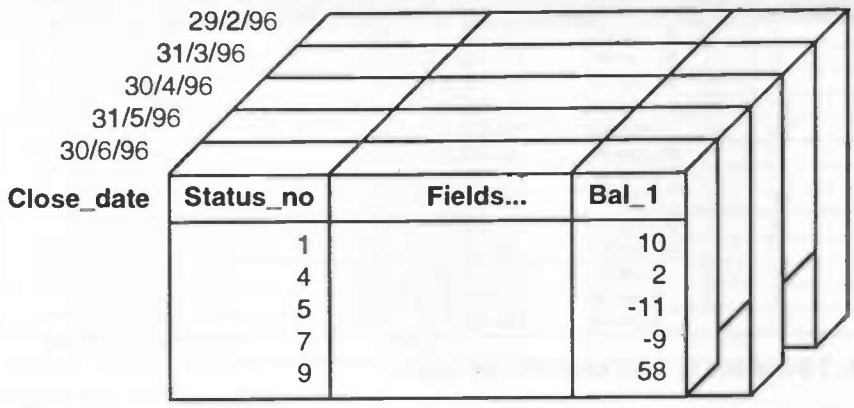


Figure 26: Concept of a FOM

Figure 27 shows the physical implementation of a FOM. The reason that there is a flag field, is that the TSOM of the closing balances might have multiple balances, in which case we need to store all those balances

Close_date	Status_no	Fields...	Bal_1	Flag1
30-Jun-96	1		150.00	T
30-Jun-96	2		200.00	T
31-Jul-96	1		376.00	T
31-Jul-96	2		875.00	T
31-Aug-96	1		340.00	T
31-Aug-96	2		213.00	T

Figure 27: FOM

A FOM is indexed by CLOSE_DATE+STATUS_NO.

To update a FOM from a SOM append the balances at the end of the month to the FOM and set the close_date field.

To update a FOM from a TSOM, a condition has to be specified, which in general will be FLAG1 .OR. FLAG2 .OR. … .OR. FLAGn, because all balances are to be appended..

Figure 28 shows the FOM entry for the inventory TSOM as described in TSOM with multiple balances on page nr. 24. In this case the first status is GODOWN, and the second is GODOWN+PRODUCT.

Close_date	Doc_ID	Doc_no	Item	Doc_date	Register	Godown	Product	Doc_value	Bal_1	Bal_2	Flag1	Flag2
30-Jun-96	33	9	1	30-Jun-96	99	GD 1	PD1	0	16	8	F	T
30-Jun-96	31	8	1	30-Jun-96	120	GD 1	PD2	0	18	8	T	T
30-Jun-96	31	3	1	30-Jun-96	58	GD 2	PD1	0	13	13	F	T
30-Jun-96	33	1	1	30-Jun-96	98	GD 3	PD1	0	12	12	T	T
31-Jul-96	33	9	1	30-Jun-96	120	GD 1	PD1	0	16	8	F	T
31-Jul-96	31	15	1	4-Jul-96	129	GD 1	PD2	0	27	19	T	T
31-Jul-96	31	3	1	30-Jun-96	58	GD 2	PD1	0	13	13	F	T
31-Jul-96	33	16	1	5-Jul-96	120	GD 2	PD2	0	20	7	T	T
31-Jul-96	33	11	1	8-Jul-96	159	GD 3	PD1	0	6	3	T	T
31-Jul-96	33	8	1	6-Jul-96	133	GD 3	PD2	0	15	3	F	T

Figure 28: FOM for the inventory TSOM

4.2.5 HOM (Historical On-line Master)

A HOM is a merge between a TSOM and a FOM. It is in fact a TSOM with the FOM entries merged into it based on the dates. The CLOSE_DATE field of the FOM is now stored in the DOC_DATE field of the HOM.

In general FOM entries have a different DOC_ID, (the FOM ID), to distinguish them from the normal documents.

Figure 29 shows the HOM entry for the inventory TSOM as described in TSOM with multiple balances on page nr. 24.

Doc_ID	Doc_no	Item	Doc_date	Register	Godown	Product	Doc_value	Bal_1	Bal_2	Flag1	Flag2
31	4	1	30-Jun-96	99	GD 1	PD1	10	10	10	F	F
51	9	1	30-Jun-96	120	GD 1	PD1	0	16	8	F	T
33	9	1	30-Jun-96	120	GD 1	PD1	-2	16	8	F	T
31	8	1	30-Jun-96	120	GD 1	PD2	8	18	8	F	F
51	8	1	30-Jun-96	120	GD 1	PD2	0	18	8	T	T
33	12	1	4-Jul-96	128	GD 1	PD2	-4	12	4	F	F
31	15	1	4-Jul-96	129	GD 1	PD2	15	27	19	T	T
51	9	1	31-Jul-96	120	GD 1	PD1	0	16	8	F	T
51	15	1	31-Jul-96	129	GD 1	PD2	0	27	19	T	T
31	3	1	30-Jun-96	58	GD 2	PD1	13	13	13	F	T
51	3	1	30-Jun-96	58	GD 2	PD1	0	13	13	F	T
31	10	1	5-Jul-96	120	GD 2	PD2	12	25	12	F	F
33	16	1	5-Jul-96	120	GD 2	PD2	-5	20	7	T	T
51	3	1	31-Jul-96	58	GD 2	PD1	0	13	13	F	T
51	16	1	31-Jul-96	120	GD 2	PD2	0	20	7	T	T
31	5	1	30-Jun-96	33	GD 3	PD1	15	15	15	F	F
33	1	1	30-Jun-96	98	GD 3	PD1	-3	12	12	F	F
51	1	1	30-Jun-96	98	GD 3	PD1	0	12	12	T	T
31	2	1	5-Jul-96	129	GD 3	PD2	8	20	8	F	F
33	8	1	6-Jul-96	133	GD 3	PD2	-5	15	3	F	T
33	11	1	8-Jul-96	159	GD 3	PD1	-9	6	3	T	T
51	8	1	31-Jul-96	133	GD 3	PD2	0	15	3	F	T
51	11	1	31-Jul-96	159	GD 3	PD1	0	6	3	T	T

Figure 29: HOM for the inventory TSOM

This HOM is displayed in the first calculation index: GODOWN+DOC_DATE+REGISTER). Of course it can also be displayed in different calculation indexes.

4.2.6 COM (Capacity On-line Master)

A COM is used to store Profile values per Capacity state. A COM is basically an extension to a POM. A third dimension is added to it, the Capacity State. This Capacity state is mostly a time-dependent state.

Figure 30 shows the concept of a COM. On the left we see the Capacity_state. Per capacity state there are different profile entries.

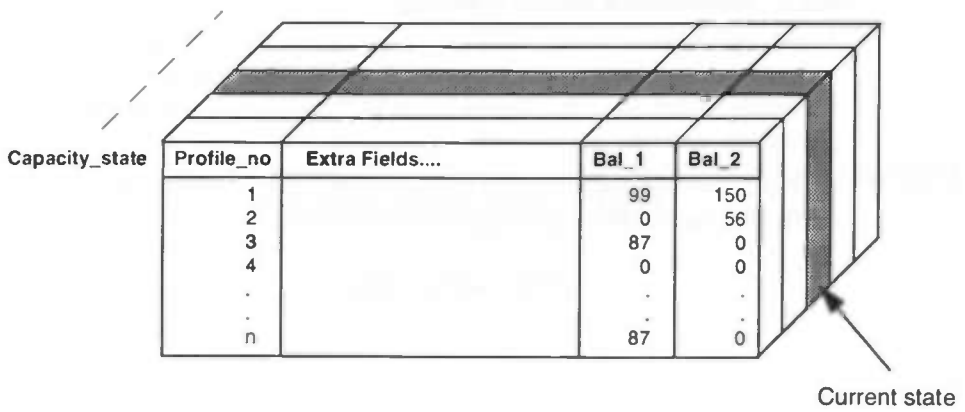


Figure 30: Concept of a COM

A typical COM is for instance an airline booking system. In this case a flight will be a Profile and the bookings throughout the different days will be the COM.

Figure 31 shows this airline COM. On the left we see the Doc_date and Flight_no which are combined the Capacity state. (Since the same flight number can appear on a different date). The seat numbers are the individual profile entries.

Doc_date	Flight_no	Seat	Customer	Price
30-Jun-96	SQ124	1	John	145.00
30-Jun-96	SQ124	2	Mary	140.00
30-Jun-96	SQ124	3	Isaac	145.00
2-Jul-96	SQ118	1	Bert	228.00
2-Jul-96	SQ118	2	Philip	230.00
2-Jul-96	SQ118	3	Albert	225.00
2-Jul-96	SQ118	4	Peter	225.00
8-Jul-96	SQ124	1	July	150.00
8-Jul-96	SQ124	2	Susan	150.00
8-Jul-96	SQ124	3	Scott	150.00

Figure 31: Airline COM

A COM is indexed by the Capacity state, which in this case is DOC_DATE+FLIGHT_NO, and the Key of the POM, which in this case is FLIGHT_NO+SEAT.

4.3 Types of posting

There are 3 types of posting the Intelligen can handle.

4.3.1 On-line Posting

This posting is as the word says online. This means that when the user keys in a document, the engine will automatically post that document into the master(s).

4.3.2 Batch Posting

When the online-posting feature is disabled, the user will have to use a batch-posting to post his documents. This batch-posting function actually uses the online-posting function by posting all the documents in a certain date-range for a certain document.

Batch posting is used to speed up document entries. This way documents can be quickly inputted during the day, and at the end of the day, these documents can be batch-posted.

4.3.3 Automated Batch Posting

Automated Batch Posting (ABP) is used to automatically post all the documents at the end of a period into a certain master. For instance at month's end closing, ABP will automatically calculate the figures brought forward and post all the documents in the period you specify.

ABP uses batch posting to post all the documents. In Figure 32 we see how ABP uses batch posting and batch posting uses on-line posting.

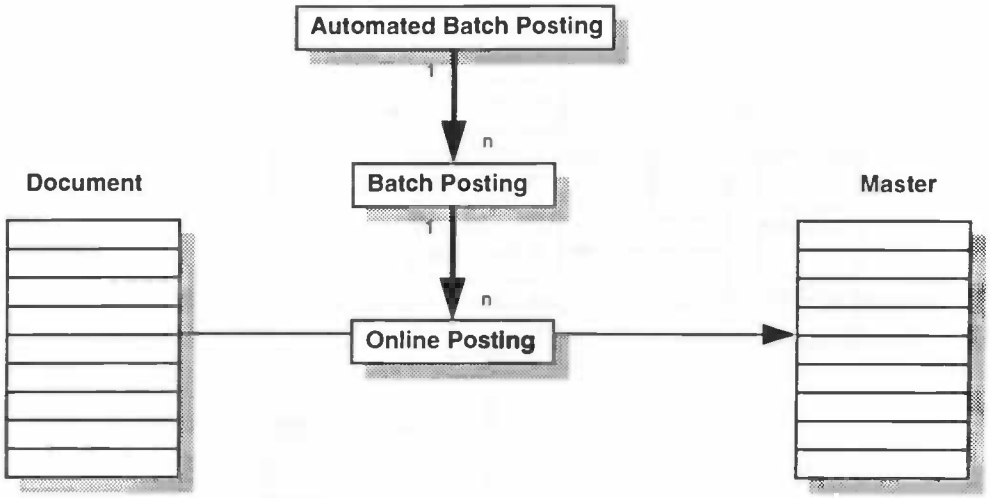


Figure 32: Posting concept of Intelligen

In detail ABP uses 3 master files: a TSOM, a FOM and a SOM. If we look at an ABP setup for a month closing, ABP uses the SOM to capture the brought forward balance of last month. It posts these balances into the TSOM. Then the documents are posted into the TSOM, and at the end the balances are calculated, which are carried forward into the SOM and into the FOM for historical rollback purposes.

Figure 33 shows the process step by step:

1. Initialize
   Create a new, empty TSOM.
2. B/F
   Post the Brought Forward (B/F) balances of the previous month into the TSOM.
3. Incoming
   Post all incoming documents.
4. Backup
   Create a backup, so that the process can start from this point in case the system hangs later on
5. Outgoing
   Post outgoing documents.
6. C/F
   Post Carried Forward (C/F) balances at the end of the month into the SOM and for historical rollback purposes into the FOM.
7. Backup
   Create a compressed backup of the TSOM.

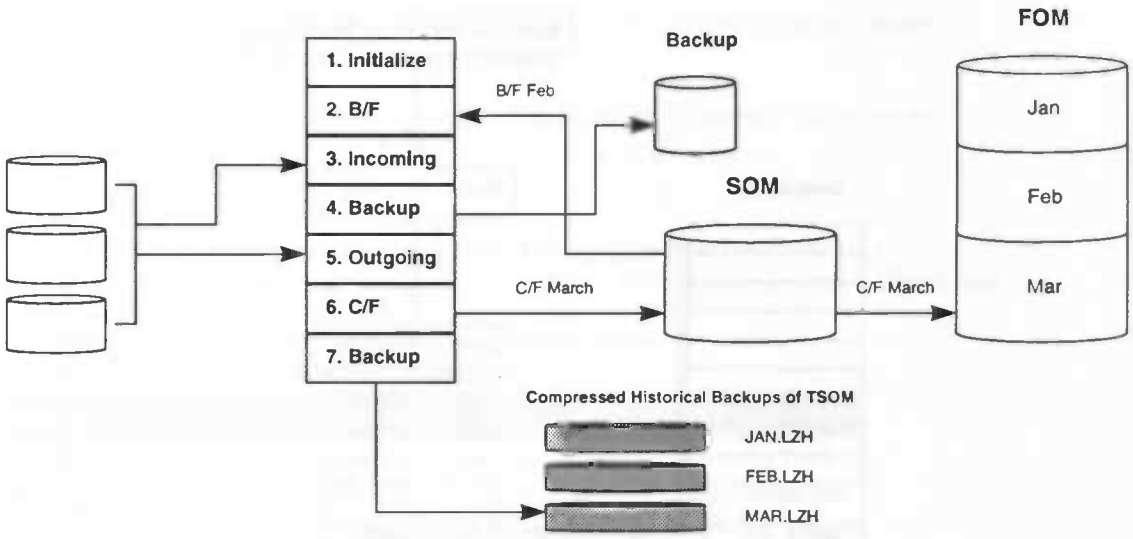


Figure 33: Procedure of ABP

5 Design

In this section we will outline the design of the batch posting engine. First we will describe relevant parameter files, Advantage specific programming and locking techniques. At the end we will give a global and functional design.

5.1 Zero-programming parameter files

The zero-programming approach uses parameter-files instead of source codes and object codes. The engine then interprets these codes directly, so no compiling and linking is necessary. The next paragraphs will briefly describe the setup of these files.

5.1.1 Document parameter files

There are 3 basic parameter-files needed to define a document, as shown in Figure 34:

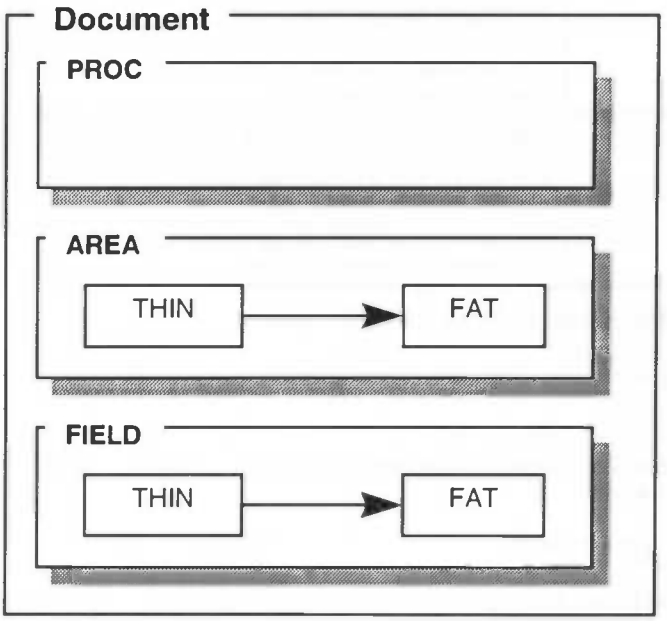


Figure 34: 3 basic parameter files of a document

5.1.1.1 PROC file

The PROC file contains data on the appearance of the document. How should it fill the screen etc. Specifically it contains the following fields:

Field	Description
Catalog entry	Name of the document as it should appear in the catalog.
Border Coordinates	Contains the values of the coordinates of the borders of the document's view. Are calculated automatically.
FAT file?	If specified as Y the Document should be viewed as a FAT file (with THIN fields in the upper parts, and a FAT-table part below). Otherwise the document should be viewed as a THIN document (centralized on the screen).
Number of fields to hide	Defines the number of (THIN) fields that should not be displayed. These fields start from the last fields in the field list, as defined in the FIELD-file. For instance, if a register-file is the last field, this field can be hidden by specifying 1 here.

Table 1: PROC-file fields

The PROC-file is a text file. The naming convention of a PROC file is as follows:

PROC*xxx*

where *xxx* specifies the document number. For instance document 312 refers to PROC file PROC312.

5.1.1.2 AREA file

The AREA file contains data in the working areas, indexes and relations of a document. If the document is a FAT file, the AREA file contains 2 parts, a THIN and a FAT part, defined by their different area number.

Specifically it contains the following fields

Field	Description
Area number	Specifies the Area number. Area 1 specifies the THIN part, Area 2 the FAT part.
DBF file name	Contains the name of the physical DBF file, in which the data should be stored.
Indexes	Contains the indexes which are used for this document. The first index is also the key, and should be used as a relation to the FAT part, if any.
Related file	If the document is a FAT file, the THIN part should contain a relation to the FAT part. For instance, if the document is 312, the THIN filename is DB312.DBF, the relation would be DB312A.DBF
Relation key	Contains the key, which should be the same as the document key, if a relation exists.

Table 2: AREA-file fields

The AREA-file is a text file. The naming convention of a AREA-file is as follows:

AREA*xxx*

where *xxx* specifies the document number. For instance document 312 would refer to AREA file AREA312.

5.1.1.3 FIELD file

The FIELD file contains the names and properties of the fields within the document. If the document is a FAT file, the FIELD file contains 2 parts, a THIN and a FAT part, defined by their area number.

Specifically it contains the following fields

Field	Description
Area number	Specifies the Area number. Area 1 specifies the THIN part, Area 2 the FAT part.
Name	Contains the name of the field
Type	Contains one letter to specify the type of the field. The following types are allowed: C - Character N - Numeric D - Date M - Memo
Length	Contains the length of the field.
Decimal	Contains the number of decimals in a numeric field

Table 3: FIELD-file fields

The FIELD-file itself is a text file. The naming convention of a FIELD-file is as follows:

FIELD*xxx*

where *xxx* specifies the document number. For instance document 312 refers to FIELD file FIELD312.

5.1.2 Posting parameter files

The Intelligen uses different kinds of parameter files. For the posting engine two kinds are relevant: Description files and Formula Files.

5.1.2.1 Description File

A description file is a file which holds information on the following items:

- What kind of append/calculation indexes are being used
- Into which master(s) should the document be posted.
- What are the calculations and calculation fields.

For each master the description file will have a different entry.

Since description files can contain references to multiple masters, we also call them Multiple On-Line (MOL) description files

5.1.2.2 Formula File

Formula files indicate which document field should be posted into which master field. It also describes for each field, what kind of field it is. There are 3 types of fields:

Input fields:	Fields that are stored on document input only.
Edit fields:	Fields that are replaced while editing source documents.
Calculation fields:	Fields that are calculated based on a calculation key specified in the description file. These kinds of fields will only appear when posting into a TSOM.

Figure 35 shows the concept of a formula file: The formula file specifies which fields of the document on the left are to be posted into which fields of the master on the right.

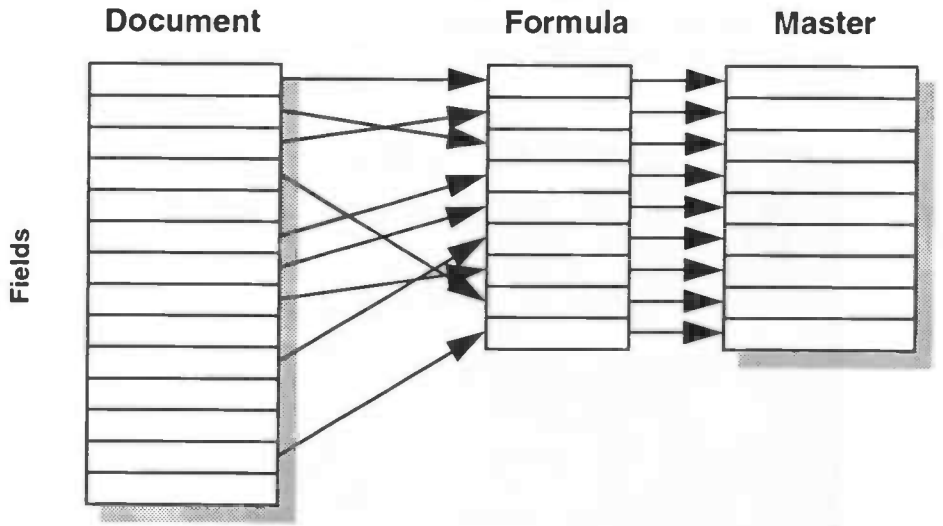


Figure 35: Concept of a formula file

Since Formula files contain reference to one Master, we also call them On-Line (OL) formula files.

The whole setup of Description and Formula files is shown in Figure 36: The description file specifies which formula files should be used to post the fields of the document into the different masters on the right.

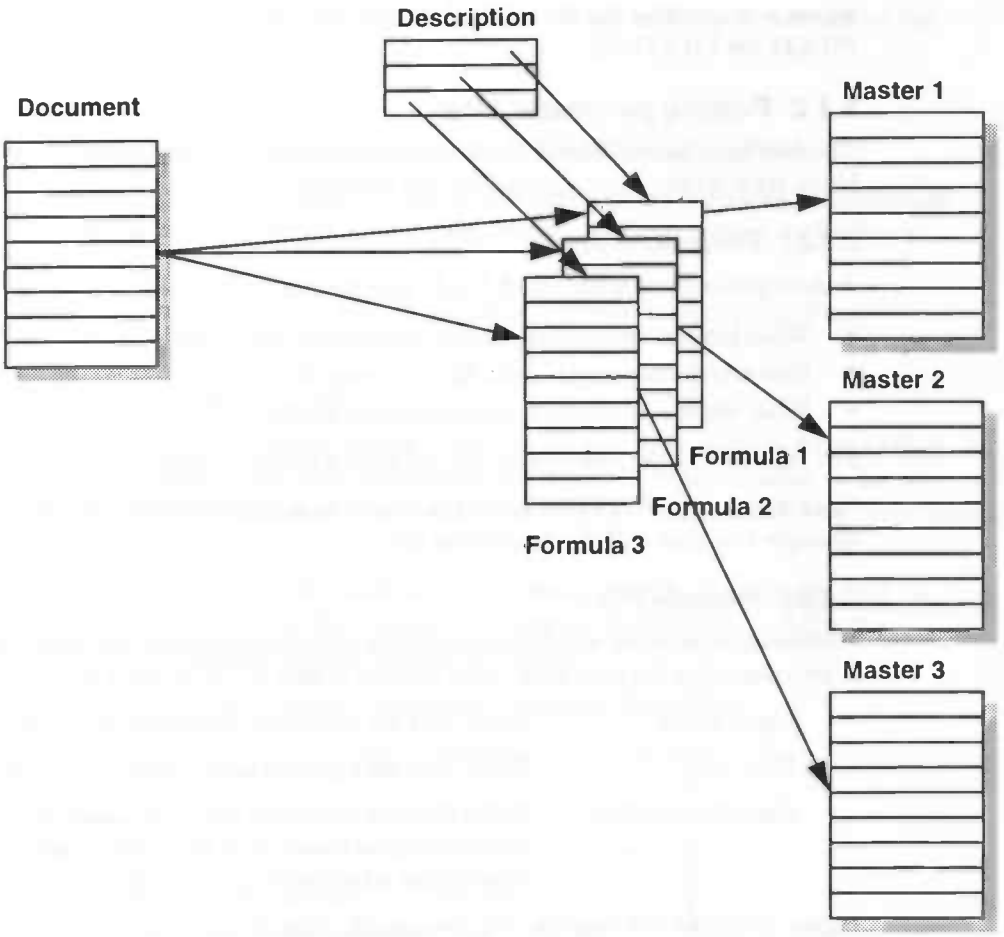


Figure 36: Setup of a Description plus Formula files

5.1.3 Automated Batch Posting parameter files

Automated Batch Posting uses parameter files as well to store information on which documents are posted into which master. The following information is stored into the files:

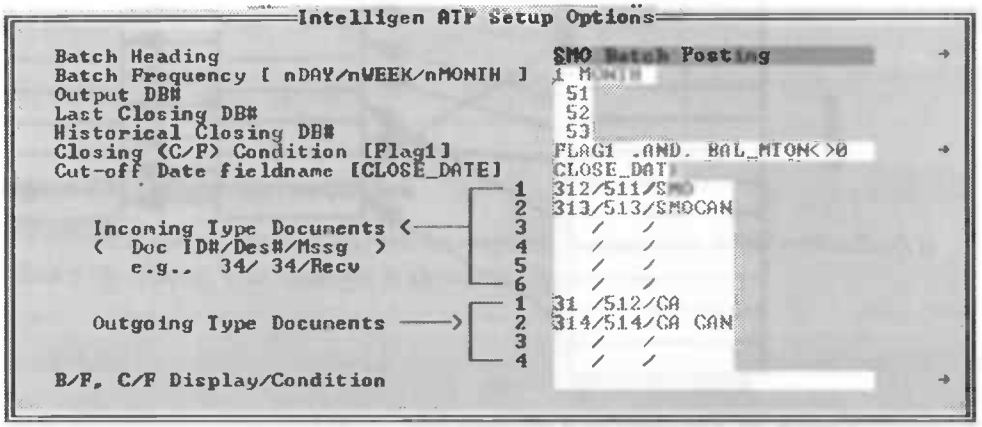


Figure 37: ABP Parameter File.

These values are then stored into a textfile. The naming convention of the ABP parameter file is as follows:

ATP*xxx*

where *xxx* specifies the document number. For instance ABP 42 refers to file ATP42. The ATP files reside in the ONLINE\ Directory.

ONLINE directory seems an odd name for batch posting parameter files, but to keep the system upwards compatible, the name was not changed.
The same goes for ATP, it used to be called Automatic Transactions Posting, but since ATP already stands for Automatic Transactions Processing, it was changed to ABP.

5.1.4 Other parameter files

There are also parameter files for the Report- and Enquiry-engine, but since we are not going to implement those, we will not go into detail on these.

5.1.5 Location of parameter files

Figure 38 gives an overview of the directories of a typical project, in this case AGRO.

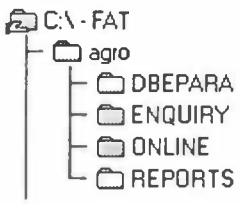


Figure 38: Directory structure of a project

The parameter files of the different sub-engines are stored in these directories. Table 4 shows which parameter files are stored where.

Directory	Parameter files
. (main)	Posting engine
DBEPARA	Database engine
ENQUIRY	Enguiry engine
ONLINE	Batch Posting engine
REPORTS	Report engine

Table 4: Location of parameter files

5.2 Advantage specific programming

There are certain functions that are not Clipper standard, but that are specific to the Advantage database server. We will briefly describe the ones used in the batch posting engine.

5.2.1 Functions

To take full advantage of the Transactions Processing System of Advantage (TPS), special functions need to be used to specify the transaction[7].

5.2.1.1 Begin Transaction

BEGIN TRANSACTION marks the beginning of a transaction

5.2.1.2 Commit Transaction

COMMIT TRANSACTION marks the end of a transaction. It signifies that all updates and appends specified during the transaction should be written to disk.

5.2.1.3 Rollback Transaction

ROLLBACK TRANSACTION marks the end of a transaction. It signifies that all updates and appends specified during the transaction should be aborted. After a

ROLLBACK TRANSACTION, the database file will be left as it was before the transaction began.

5.2.1.4 AX_Transaction()

AX_Transaction([<nTransactionState>]) -> lCurrentlyInTransaction

can be used to begin, commit or rollback a transaction, by supplying an additional nTransactionState. (1=Begin Transaction, 2=Commit Transaction, 3=Rollback Transaction).

If no parameter is passed to AX_Transaction(), then a .T. is returned if user is currently in a transaction and an .F. otherwise.

5.2.1.5 AX_TPSCleanup()

AX_TPSCleanup(<cServerName>) -> Lsuccess is used to recover from a failed transaction. When called with the network file server and volumename as the only parameter, AXS_TPSCleanup() will perform a failed recovery, which may occur after a system crash.

Before a transaction is performed, all the necessary records should be locked. No other records should be locked or unlocked during the transaction. After the transaction, the records should be unlocked again.

Figure 39 shows how a general transaction sequence should look.

```
Function Do_transaction()
/* Lock all necessary records
*/
Begin Sequence
   Begin Transacion
      /* Transaction */
   Commit Transaction
   Recover using oError
   lSuccess := .F.
End Sequence
Return lSuccess
```

Figure 39: Transaction sequence

If a run-time error occurs during the transaction, the function TPSErrHandler() is called with oError. This function is shown in Figure 40.

```
Function TPSErrHandler( oError )
If ( AX_Transaction() )
   // We are in a transaction. Roll it
back.
   Rollback Transaction
Endif
// Transfer control to the RECOVER
statement in
// the Order Processing Routine
Break oError
Return NIL
```

Figure 40: TPSErrHandler() function

5.2.2 Programming concepts

There are some programming techniques that are applied differently in a client/server architecture than in a central host[8].

5.2.2.1 File handles

If in a client/server environment, multiple users access the same database file, only one file-handle will be opened, while in a central host architecture the file handles are equal to the number of users that want to access the file. So this greatly reduces the number of open files.

The down side is that because of the bigger overhead, opening and closing files will take slightly longer. So in general all the files should be opened in the beginning and closed in the end and should not be closed or reopened in between.

5.2.2.2 When and how to use transactions

Writing data to a single database file should not be considered as a transaction. Only if data is supposed to be written to multiple database files, one should use transactions. So in the Intelligen's case, posting of data is considered a transaction.

Transactions should be limited in their complexity. In most applications, the only operations that should be included within transactions are insert, update and delete operations. More complex transactions will slow down multi-user performance.

With the Intelligen this leaves the question what to define as a transaction. Since after posting a record into the master, the master has to be recalculated from that point, in my opinion there are three ways of defining a transaction:

- A single record update into the master
- Update of one master (including recalculation)
- Update of all the masters as defined in the description file.

We have chosen a transaction to be defined as the update of one master. If it is defined as the update of a single record in a master, the data might only be partially calculated, which is wrong. If it is defined as the update of all the masters as specified in the description file, record locks on different masters will be kept too long, which will restrict other users from accessing that data.

5.2.2.3 Use of indexes

There are 3 types of indexes:

- Non-compact indexes
  Non-compact indexes require more disk space and are potentially faster when accessing small files or when many modifications or updates may occur
- Compact indexes
  Compact indexes save disk space and may increase the speed of an operation when accessing large data sets. A compact index can be up to 90% smaller than an equivalent non-compact index.
- Compound indexes
  A compound index is a single index that contains multiple indexes. Compound indexes allow for several indexes in the application while only using one file handle.

Since most of I&J's customers use really large database-files, we have chosen to use compact indexes. Compound indexes are not used for backward compatibility reasons.

5.2.3 Configuration of the server

In order for the server to function optimally, some changes need to be made to the standard configuration. The database server is automatically configured by the AXS.CFG file at startup. See also Appendix A: AXS.CFG Server configuration file on page nr.60.[9]

We will only briefly describe the items we have changed from the default values.

In order to configure some settings, like User Buffer Size, Burst Mode Packets and Number of Receives the maximum record size has to be known. We checked with some large records in existing systems, and the biggest size we could find was approximately 600 bytes, so there is no need to adjust these parameters.

5.2.3.1 Number of Worker Threads

This is the number of worker threads used to service client database file requests. Adjusting the number of worker threads controls CPU utilization of the database server. Range = 1 - 16. Default 3.

We changed this value to 8, since there are no other tasks running on the server.

5.2.3.2 Number of Data locks

This is the total number of locks that can be obtained on all the open database files. A file-lock and a record-lock are all considered as one lock. For the use of record-locking, the number of locks has to be increased, since locks need to be obtained for all the recalculation records, see Locking methods on page nr. 39.

We changed this value to 250.

5.3 Locking Techniques

To ensure serializability data files should be locked according to certain procedures[10].

5.3.1.1 Granularity of data items

Granularity of data items indicates what portion of the database an item represents. Advantage can lock data in 2 granularities:

- A whole file
- A database record

5.3.1.2 Types of locks

There are 2 types of locks, shared and exclusive locks. If a user has a shared lock on an item, other users will have read-only access to that item. If a user has an exclusive

lock on an item, other users cannot access that item. Advantage supports the following options:

- A complete file - Shared or exclusive
- A database record - Exclusive only

5.3.1.3 Locking methods

Since the all records have to be locked before beginning a transaction[11], the locking is performed according to the conservative 2 phase locking method. Conservative 2 PL requires that every transaction locks all the items it needs in advance; if any of the items cannot be obtained, none of the items are locked. Rather, the transaction waits and then tries again to lock all the items.

The good thing about conservative 2 PL is that it is dead-lock free[12].

The Advantage manual gives a standard LockRecords() function[13], but since it is designed for a client/server area, it is not very cpu-time friendly, so we have rewritten this function that it waits a while before locking.

5.4 Global design

5.4.1 Modules

The program is split into 3 modules, see Table 5:

Module	Description
ABP.PRG	This module contains the main functions of the program.
IG.PRG	This module contains more general functions which are used by other engines as well and is later converted to a library
IGKEY.PRG	This module contains the code to control the amount of processor time used by the program while in idle mode.

Table 5: Modules

5.4.2 Data structures

5.4.2.1 Structure of the MOL-description file

The MOL (Multiple Online) description files are all plain text files. Figure 41 shows the structure of the file: per master there is a line with the transaction- and appendkey and subsequent lines with a calculation key, followed by a "~" and the calculation fields corresponding to that calculation key.

```
<Master_1>
<Transaction_key>|<Append_Key>
[<Calc_key_1>~<Calc_1_field_1>,…,<Calc_1_field_n>]
…
[<Calc_key_n>~<Calc_n_field_1>,…,<Calc_n_field_n>]
…
…
[<Master_n>
<Transaction_key>|<Append_Key>
[<Calc_key_1>~<Calc_1_field_1>,…,<Calc_1_field_n>]
…
[<Calc_key_n>~<Calc_n_field_1>,…,<Calc_n_field_n>]]
```

Figure 41: Structure of a MOL-description file

Figure 42 shows an example of a typical MOL description file.

```
42
" 31" + DOC_NO|" 31"+DOC_NO+STR(ITEM,2)
CA_NO ~ BAL_MTON, BAL_QTY, BAL_VAL
46
" 31" + DOC_NO|" 31" + DOC_NO + STR(ITEM,2)
SMO_NO ~ SMO_DATE, UT_SPRICE, SMO_MTON, CAN_MTON,
CA_MTON, BAL_MTON, BAL_QTY, BAL_VAL
45
" 31" + DOC_NO|" 31" + DOC_NO + STR(ITEM,2)
GODWN_CODE + PROD_CODE~BAL_MTON
GODWN_CODE + PG_CODE~GBAL_MTON
```

Figure 42: Example of a typical description file

5.4.2.2 OL Formula files

The OL (On-Line) formula files are also plain text files. They contain the field names of the Master plus a description of the contents to fill the master with. Figure 43 shows an example of a typical formula file.

```
DOC_NO        DB32->DOC_NO
ITEM          ED_FILE->ITEM
DOC_DATE      ~DB32->DOC_DATE
REGISTER      DB32->REGISTER
SYS_DATE      DATE()
SYS_TIME      SECONDS()
CA_NO         ~DB32->CA_NO
GODWN_CODE    ~DB32->GODWN_CODE
PORT          @PREV[10]
```

Figure 43: Example of a typical Formula file

The description fields can be preceded by the following:

(blank) - The field is a REPLACE field, which means that it is only updated the first time a document is filled out.

~ - The field is an EDIT field, which means it is always updated upon Input or Edit.

@ - The field is an UPDATE field, which means that the contents are recalculated every time a record within the calculation range is updated.

5.4.2.3 Structure of MOL-array

All these Description and Formula files are read into memory at startup of the application, so reference to it at a later time is much faster. This array is called the MOL-array. Figure 44 shows the structure of the MOL array.

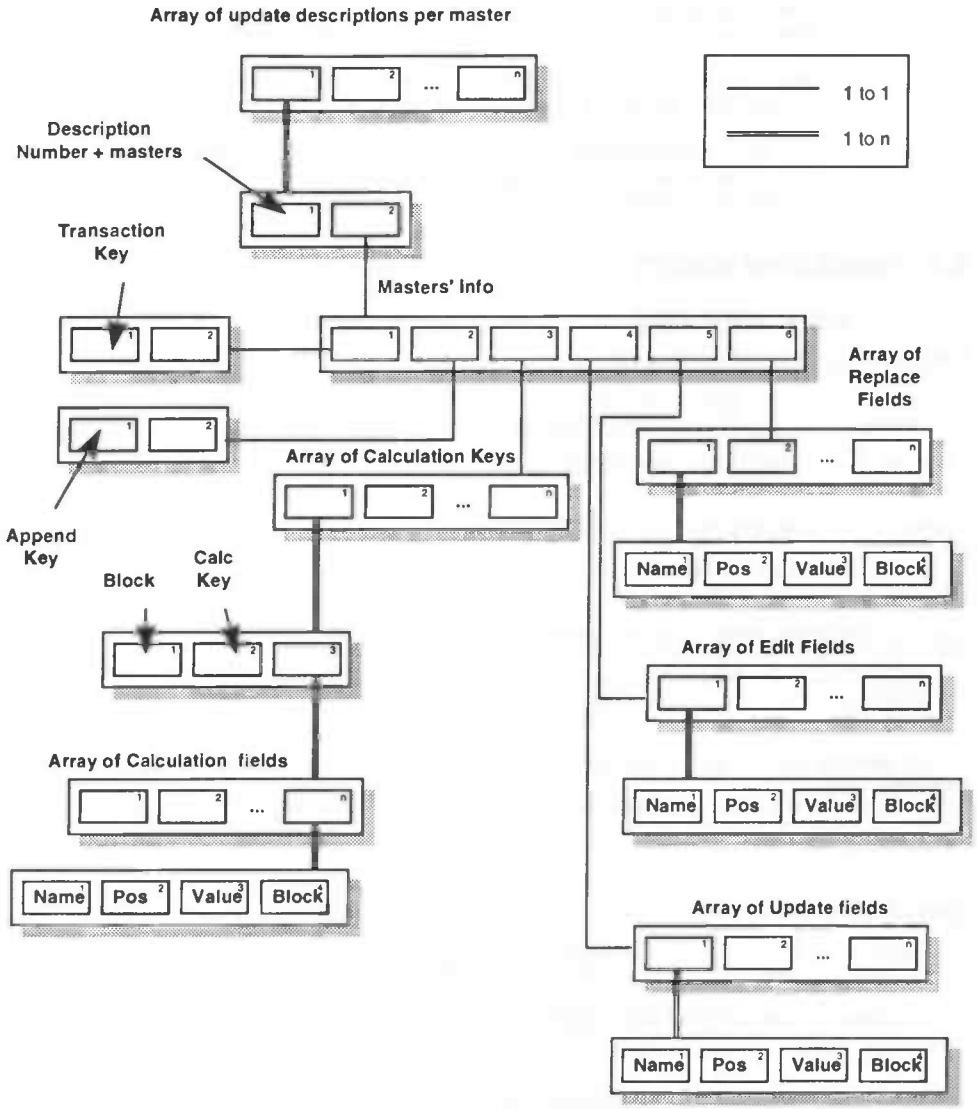


Figure 44: Structure of the MOL array.

The main difference between the Update and Calculation fields is that the Calculation keys can be multiple sets depending on the number of calculations[14], whereas the Update fields are all the Calculation keys combined.

To reference these fields quickly without having to go into too many array references, the following shortcuts are implemented:

```
#define SKEY    1
#define INFO    2

#define XKEY    1               //      Transactions
#define AKEY    2               //      Append
#define CKEY    3               //      Calculations
#define RFRM    4               //      Replace formulae
#define EFRM    5               //      Edit    formulae
#define CFRM    6               //      Calc    formulae

#define KEY             1
#define BLK             2
#define FLD             3

#xtranslate     _xKEY   =>      aMol\[ nPos, INFO, XKEY, KEY \]
#xtranslate     _xBLK   =>      aMol\[ nPos, INFO, XKEY, BLK \]
#xtranslate     _aKEY   =>      aMol\[ nPos, INFO, AKEY, KEY \]
#xtranslate     _aBLK   =>      aMol\[ nPos, INFO, AKEY, BLK \]
#xtranslate     _CALC   =>      aMol\[ nPos, INFO, CKEY \]
#xtranslate     _RFRM   =>      aMol\[ nPos, INFO, RFRM \]
```

```
#xtranslate     _EFRM   =>      aMol\[ nPos, INFO, EFRM \]
#xtranslate     _CFRM   =>      aMol\[ nPos, INFO, CFRM \]

#xtranslate     Get_aMOL( <cKey> )      =>      ;
  ASCAN( aMOL, { |a| LEFT( a\[ SKEY \], 7 ) == <cKey>  } )

// Get_aMAS(cKey). Scan aMOL for master cKey
#xtranslate     Get_aMAS( <cKey> )      =>      ;
  ASCAN( aMOL, { |a| LEFT( a\[ SKEY \], 3 ) == <cKey>  } )
```

5.5 Functional design

5.5.1 ABP.PRG

```
FUNCTION igATP( xID, cMono )
/*
 * xID          - Description file
 * cMono        - Monochrome display?
 *
 * Runs the Intelligen Automated Batch Posting
 */

STATIC FUNCTION ATP_New( )
/*
 * Creates a new Parameter file.
 */

STATIC FUNCTION GSet_Status( xNewValue, nOffset, lWRITE )
/*
 * xNewValue    - New value to store into static array xValue
 * nOffset              - Offset to store xNewValue
 * lWrite               - Write to file?
 *
 * If xNewValue is an array and nOffset is empty, writes xNewValue to the static array
xValue.
 * Otherwise write xNewValue at offset nOffset into xValue.
 * If lWrite, write to file.
 * Returns old value
 */

STATIC  FUNCTION        Displ_Status( nWait, cHead)
/*
 * nWait                - Number of seconds to display the box, 0 is infinity
 * cHead                - Heading for box
 *
 * Display a status box for nwait seconds (0 is infinity), with heading cHead
 *
 */

STATIC  FUNCTION        RW_Status( aStatus )
/*
 * aStatus      - Array in which to store the status
 *
 * If aStatus is empty, read in array from file STAT<cId> in the ONLINE directory
 * e.g. STAT42. Otherwise, write aStatus to that same file. Returns xValue (= current
 * status)
 */

STATIC FUNCTION Help_ATP( )

STATIC FUNCTION ChangeDate( cLastCut, cCut, cDate, bDate, nFreq )
/*
 * cLastCut     - Filename of last cut date     (Work FOM)
 * cCut         - Name of historical cut                (FOM)
 * cDate        -
 * bDate
 * nFreq        - frequency, 300 for monthly
 *
 * Prompts the user for a new closing date. If earlier postings exist, it will pop up
a list
 * of earlier closing dates.
 */

FUNCTION POST( nXsac, nDES, cFor, lCalc, cDATE_FIELD, lPromptDate, lPause, lNoRange,
lThin )
/*
 * nXsac                - Document nr.
 * nDes                 - Description File number
 * cFor                 - For condition
 * lCalc                - Post & Calculate?
 * cDate_field  - Date field in document
 * lPromptDate  - Prompt for date range?
 * lPause               - Pause after posting?
 * lNoRange             -
```

```
 * lThin                - Post THIN only?
 *
 *
 */

STATIC FUNCTION DispCount( nRow, nCol, nCount, nWhen )
/*
 * nCol          - Column
 * nRow          - Row
 * nCount        - Number to display
 * nWhen         - Display every
 *
 * Displays nCount at nCol, nRow every nWhen
 */

STATIC FUNCTION Express_Struc( aStruc, aFieldPos )
/*
 * aStruc               - Array that contains the structure of the database
 * aFieldPos    - Array of field numbers to convert
 *
 * Converts the fields in aFieldPos into strings, and concatenates them with "+"
 * e.g. field 1 = DOC_DATE and field 2 = REGISTER: {1,2} ->
DTOC(DOC_DATE)+STR(REGISTER,9)
 */

STATIC FUNCTION iSAY( something, atLine )
/*
 * something    - something to say
 * atLine               - at line nr.
 *
 * Return an array of something to say at which coordinates plus the stored previous
 * screen
 */

FUNCTION iCLS( aS )
/*
 * aS   - Restore array
 *
 * Restore screen according to aS
 */


FUNCTION        INIT_MOL( )
/*
 * Initializes the MOL array. For details of the MOL-array structure, see
 * Redmer Schukken: "The Intelligen 3-tier system".
 *
 * Init_MOL() function is designed to speed up all update operations
 * in a multi-user multi-tasking network. Whenever a user logs in to a
 * network to run an application, the application loader in this case
 * the MENU.EXE will pre-load all update descriptions for the current
 * session and set it up through a get/set function accessible to all
 * online or batch posting functions. ( i.e. MOL(), POST() or CALC()
 * functions. )
 * Pre-loading of descriptions come from the source file, MOL<DesNo>.DES
 * and its associated OL<DesNo>_<MasNo>.ARR files.
 */


FUNCTION        Set_aMOL( cFile )       // No Extension pls.
/*
 * cFile       - Parameter File
 *
 * Set_aMOL( ) will accept either an update description file like
 * MOL31 or just an Online formula file like OL31_41
 * The function will update only those formulas associated with the .DES
 * or .ARR file
 */

STATIC  FUNCTION        Build_SMOL( cFile )
/*
 * cFile - Description file name
 *
 * Reads the MOLarray structure as described in Redmer Schukken: "The Intelligen
 * 3-tier system", and returns it.
 */

STATIC  FUNCTION        Read_Desc( cFile )
/*
 * cFile - Filename of the description file to read
 *
 * Reads the description file into an array and returns it
 * The structure of the array is as follows:
 *
 * {{<Master_1>,{<Transaction_key>+<Append_Key>,;
 *               <Calc_key_1>+<Calc_1_field_1>+.+<Calc_1_field_n>,;
```

```
 *                .
 *                <Calc_key_n>~<Calc_n_field_1>+.+<Calc_n_field_n>}},;
 *    .,;
 *    <Master_n>,{<Transaction_key>+<Append_Key>,;
 *                <Calc_key_1>+<Calc_1_field_1>+.+<Calc_1_field_n>,;
 *                .
 *                <Calc_key_n>+<Calc_n_field_1>+.+<Calc_n_field_n>}}}
 */

STATIC  FUNCTION ReadLine( cLine )
/*
 * cLine - Line to read
 *
 * Reads cLine and determines if there is a ":" in the line. If so
 * it returns the substring after the ":", otherwise it returns cLine
 */

FUNCTION RidSign( cFrml )
/*
 * cFrml        - Formula to strip
 *
 * Strip '~', '@' and ' ' from cFrml
 */


FUNCTION MOL( nDes, cType, lRecordLocking, lFileLocking )
/*
 * nDes                         - Description File Number
 * cType                        - "ADD", "EDIT" or "DEL"
 * lRecordLocking       - Use record locking instead of file locking? Default .T.
 * lFileLocking         - If not lRecordLocking, use File Locking? Default .T.
 *
 * Online posting engine.
 */

STATIC FUNCTION Post_Mas( nDes, cMas, cType, lRecordLocking, lFileLocking )
/*
 * nDes                 - Description Number
 * cMas                 - Master Number
 * cType                - Type of Master
 * lRecordLocking       - Use record locking instead of file locking? Default .T.
 * lFileLocking         - If lRecordlocking is .F., lock and unlock file? Default to
.T.
 *
 * Posts according to nDes for Master cMas.
 */

STATIC FUNCTION DelSkip( n )
/*
 * n
 *
 * If n is NIL return value of next record, or -1 if eof().
 * If n<0 goto eof()
 * Else goto(n)
 */

STATIC FUNCTION Post_Item( cType, nPos )
/*
 * cType        - Type of posting (DEL/INS)
 * nPos         - Position of Description + Master in the aMol array
 *
 * Post the current record in the document and update the calculation
 * fields, if any.
 */

STATIC FUNCTION Get_Previous( bCalc, vCalc, cFlag, CalcNo, lScatter )
/*
 * bCalc        - Calculation key in block form
 * vCalc        - Current calculation key value
 * cFlag        - if cFlag=="N" update FLAG value with .N., .T. otherwise
 * CalcNo       - Calculation Number to look for
 * lScatter     - Fill PREV and return array?
 *
 * Get the previous values for calculation number CalcNo. A previous value is
 * the value that is generated when the index is set to the CalcNo index and skip -1
 * if within the same calculation value. Otherwise the value is empty.
 * if lScatter returns an array containing the previous values and stores the previous
values
 * in the public array PREV.
 * Updates the FLAG no. which belongs to the CalcNo to cFlag.
 */


STATIC FUNCTION Trash_it( )
/*
 * Trash current record by emptying all the fields and flagging it as deleted,
```

```
 * so it can be recycled.
 * Return 0 if next record is eof() otherwise return next record position.
 */

STATIC FUNCTION GotoBlank()
/*
 * If the first record is blank, recycles the first record by marking it as undeleted
 * and sets the pointer to it. Otherwise appends a blank record
 * Returns .T. if succesful, .F. otherwise
 */

STATIC FUNCTION ListAsArray( cList, cDelimiter )
/*
 * cList                - String to convert to an array
 * cDelimiter   - Delimiter between elements
 *
 * Converts cList, delimited with cDelimiter to an array and returns it.
 */

FUNCTION Gset_Engine( xNewValue, nIndex )
/*
 * xNewValue     - New value to store
 * nIndex        - At what index position
 *
 * Sets xNewvalue at index position nIndex and stores it. If parameters
 * are omitted, returns the Oldvalue.
 */

STATIC FUNCTION TPSErrorHandler( oError )
/*
 * oError        - ErrorType
 *
 * This error handler replaces the default Clipper error handler. If a runtime error
 * occurs, this routine will automatically be called by the Clipper engine. This
routine rolls
 * back the transaction, if a transaction is in progress. Control is then transferred
 * to the position where the error occurred
 */
```

5.5.2 IG.PRG

```
FUNCTION Net_Use( search, lNonshared, cDBAlias, lReadOnly, wait, cDriver)
/*
 * search               - file [path]name
 * lNonshared   - Open exclusive? Default .f.
 * cDBAlias             - Open as Alias? Default dbname
 * lReadonly    - Open as readonly? Default .f.
 * wait                 - number of seconds to try to open database
 * cDriver              - Driver to use. Default DEFAULT_ADVANTAGE_DRIVER
 */

FUNCTION fil_lock( wait )
/*
 * wait - number of seconds to wait
 *
 * Tries to lock current area file for wait seconds. Returns .T. if obtained,
 * .F. otherwise
 */

FUNCTION net_add( wait )
/*
 * wait - number of seconds to try to append
 *
 * Tries for wait seconds to append a blank record to the current working area,
 * if wait=0 forever, wait is default 8
 */

STATIC FUNCTION UserInfo( Otype, fname, mode, wait )
/*
 * Otype                - Message number
 * fname                - File name
 * mode         - Shared/Exclusive
 * wait         - How many seconds
 *
 *  Display Message to user for waiting to obtain locks.
 */


FUNCTION Open_alias(cDBpathname, cExt, lNonShared, cDBAlias, lReadOnly, lNtx, cDriver,
lNetpath )
/*
 * cDDBpathname         - Pathname of database to open
 * cExt         - Extension
 * lNonShared   - Open exclusive? default .f.
 * cDBAlias             - Open as alias. default dbName
 * lReadOnly    - Open as readonly? default .f.
 * lNtx         - Open with index file(s) name if character and set index to cNtx.
```

```
 *                        Or reset indexes if boolean
 * cDriver              - Driver to use, Advantage or Clipper.
 *                          Default to DEFAULT_ADVANTAGE_DRIVER
 * lNetPath             - Respect the NET_PATH?
 *                        Default to .T. if cExt=='DBF' otherwise
 *                        Default to .F.
 * Opens database cDBpathname, if not already open. Otherwise, sets the current
 * work area to it.
 * Returns .T. if the workarea exists, otherwise .F.
 */


FUNCTION Occurs( cSearch, cTarget )
/*
 * cSearch      - string to search for
 * cTarget      - string to search in
 *
 * Returns the number of times cSearch occurs in cTarget
 */

FUNCTION MakesureDir(Dirname)
/*
 * Dirname      - Directory Name
 *
 * Check if directory Dirname exists. If not, create Dirname.
 * Return .T. if created, .F. otherwise
 */

FUNCTION GetSetEnv( cMode )
/*
 * cMode        -
 *
 * if cMode is empty, restores the last pushed environments. Otherwise it pushes cMode
 * onto the environment stack.
 */

FUNCTION Save_Arr( the_array, filename )
/*
 * the_array    - Array to be saved
 * filename             - Save as filename
 *
 * Tries to save the_array as a text file for 4 seconds.
 * Returns .T. if succesful, .F. otherwise
 */

FUNCTION Rest_Arr( filename, lscr )
/*
 * filename     - File name to put into an array
 * lscr         - Display "."'s on screen while loading. Default to .F.
 *
 * Puts the contents of filename into an array and returns it.
 */

Static FUNCTION         SaveArr(the_array)
/*
 * the_array    - array to save
 *
 * Saves the_array into a file. If the_array is an array of arrays, save each array
 * as separate files.
 */


Static FUNCTION         Restarr()
/*
 * Restores array???
 */


FUNCTION Glob_index( xdBe, lprompt, lunique, lWait )
/*
 * xdBe         - Database number to index
 * lprompt      - Prompt afterwards?
 * lunique      - Remove duplicates for first index?
 * lwait        - Wait after indexing each file?
 *
 * Indexes the database no. xdBe. If xdBe is empty, indexes all databases as specified
in
 * Catalog.cat.
 * If xdBe is numeric, indexes that file. If it is a string, indexes the files in the
string,
 * separated by commas.
 */

FUNCTION TMENU( aMenu, nRow, nCol )
/*
 * aMenu        - Array of menu options
```

```
 * nRow         - Number of Rows
 * nCol         - Number or Columns
 *
 * Creates a Pop-up menu with aMenu as menu items and size nCol x nRow
 * Returns the position of the selected item.
 */

FUNCTION DIALOG( cMsg, aPrompt, cColor )
/*
 * cMsg         - Message to display
 * aPrompt      - Array of buttons to prompt
 * cColor       - Color
 *
 * Displays a dialog box with buttons as specified in aPrompt. Returns the offset of
the button
 * in the aPrompt array, the user selects.
 */

FUNCTION ARREAD( cFile, nLines )
/*
 * cFile        - Filename to read
 * nLines       - Maximum number of lines to read
 *
 * Reads cFile into an array up until the maximum nLines.
 * Each new line is a new entry in the array.
 */

FUNCTION Str_Parse( cToParse, cDelim, nlen )
/*
 * cToParse - String to parse
 * cDelim   - Delimiter
 * nLen     - Length
 *
 * Returns cToParse if cDelim is not found in cToParse, otherwise returns leftstring
of
 * cToParse until cDelim (not including cDelim)
 * Updates cToParse if passed by reference (@). if not found cToParse:="" otherwise
 * cToParse:= substring from delimiter on to nlen.
 */

FUNCTION Pops( msg, delay, ret )
/*
 * msg  - message to display
 * delay        - duration of display
 * ret  - Return
 *
 * Display msg in a pop-up box on screen, for delay time. If delay is omitted, until
 * next key-stroke. Return ret.
 */

FUNCTION Uscreen( msg, no_blink )
/*
 * msg  - Message to print
 * no_blink     - force no blinking on screen for screens with less colors
 *
 * Print message on screen.
 */

FUNCTION Get_reply( msg, mvar, pic, cval )
/*
 * msg  - Message to display
 * mvar - Memory variable to get
 * pic  - Picture for mvar
 * cval - Validation function
 *
 * Displays a diaglog-box with msg as message, and ask for input of mvar with picture
 * pic, and validates it with cval. Returns mvar.
 */

FUNCTION zero(ndividend, ndivisor)
/*
 * Divides ndividend by ndivisor. Returns 0 if ndivisor==0, result otherwise
 */

FUNCTION elapsetime( cStartTime, cEndTime )
/*
 * cStartTime  - Starting time
 * cEndTime             - Ending time
 *
 * Returns the difference between the two, or if end time is smaller than begin time
 * returns 86400
 */

function FilePath( cFile )
/*
 * cFile - File name
```

```
 *
 * Returns the path of a filename. e.g. if cFile="C:\TEST\DB1.DBF" the function
 * will return "C:\TEST\"
 */


function FileBase( cFile )
/*
 * cFile - File name
 *
 * Returns the filename of a file. e.g. if cFile="C:\TEST\DB1.DBF" the function will
 * return "DB1"
 */

function FileExt( cFile )
/*
 * cFile - File name
 *
 * Returns the extension of a filename.
 */

FUNCTION Video_push()
/*
 * Pushes the video information (cursor shape, row, column and colour) onto stacks.
 */

FUNCTION Video_pop()
/*
 * Pop Video settings (Cursor shape, position and colour) and apply it. Return nil
 */


FUNCTION Shadow( t,l,b,r )
/*
 * Creates shadow for coordinates t,l,b,r,
 */

FUNCTION StackNew()
/*
 * Create a new stack
 */

FUNCTION StackPush( aStack, exp )
/*
 * aStack       - Stack to push element on
 * exp          - element to push
 *
 * Add new element to the stack array and then return the array
 */

FUNCTION StackPop( aStack )
/*
 * StackPop( <aStack> ) --> value
 * Pop a value from the stack
 *
 * return NIL if nothing is on the stack.
 */

FUNCTION pBox( aDesc, aVals, aPic, aValids, cHead, aWhen, nTop, nLeft )
/*
 * aDesc        - Array of Description values to display
 * aVals        - Array of values to get
 * aPic - Array of pictures for the values
 * aValids      - Array of post-validations for the values
 * cHead        - Header for the box
 * aWhen        - Array of pre-validations
 * nTop - Top coordinate
 * nLeft        - Left coordinate
 *
 * Display a pop-up box with aDesc as headers, aVals as value to get, aPic
 * as pictures, aValids as validation checks, cHead as header for the box, aWhen
 * as Prevalidation functions at nTop and nLeft.
 * Returns the values of aVal.
 */

FUNCTION TimeAsString( nSeconds )
/*
 * nSeconds     - Time to present
 *
 * Return the time as specified in nSeconds as a string
 */

FUNCTION Init_db( xId )
/*
 * xId - DB number
 *
```

```
 * Initializes database xId, considering the net_path
 */

FUNCTION Init_NETPATH()
/*
 * Get the netpath from IG.INI if file exists
 */

FUNCTION Open_dBe()
/*
 *      Opens the parameter files as specified in the static variables, cProc, cArea
and
 *      cDat.
 *      Extits with cArea as the current work area.
 */

FUNCTION Sys_path( nArea, Fpath, Fname )
/*
 * nArea        - Area number
 * Fpath        - File path
 * Fname        - File name
 *
 * Return the system path as defined in IG.INI and return the path+dbfname
 */

FUNCTION Get_struc( nArea )
/*
 * nArea        - Area number
 *
 * Restore the structure of the dbf file as defined in the static variable cDat, into
the
 * static variables Field...
 */

FUNCTION Chk_Dbf( )
/*
 * Check if the current open database as specified in DBF_NAME, has the same structure
as defined
 * in the static variables field... Return "SAME" if it is the same, "DIFF" it is
different or
 * "MISS" if the file is missing.
 */

FUNCTION Make_dbf( carry_data, nArea )
/*
 * carry_data  - carry data?
 * nArea                - Area number
 *
 * Restore structure of DB_NAME to the structure as specified in the variables
field...
 * If carry_data==[DIFF], carry data from old file. Otherwise just create a new file.
 *
 */

FUNCTION Make_ndx( c, lunique, lWait )
/*
 * c            - Area number
 * lunique      - Remove duplicates for first index?
 * lWait        - Wait afterwards?
 *
 * Deletes old indexes and Creates the index file(s) as specified in the cArea
parameter.
 */

STATIC FUNCTION BetterIndex( cIndex, ntx_path, lDescend )
/*
 * cIndex       - Index key
 * ntx_path     - file name
 * lDescend     - Index in descending order?
 *
 * Indexes the file ntx_path on cIndex. If lDescend in descending order. Using
 * The standard database driver. Returns .t. if no errors occur, otherwise
 * it returns .f.
 */

STATIC  FUNCTION Express( fld_exprs )
/*
 *      returns properly expressed field expressions
 *      Use of PUBLIC F_NAME,F_TYPE,F_LEN,F_DEC
 *      e.g.,
 *
 *      NAME + STR(INV_NO,8)   -> NAME + STR(INV_NO,8)
 *      LOCATION + TIME                -> LOCATION + STR(TIME,6)
 *      P_CODE+DESCEND( CREATED + TIME )
 *      ->      P_CODE+DESCEND( CREATED + TIME )      // Unchanged!
 */
```

```
FUNCTION BuildRecArray(aRecords)
/*
 * aRecords     - Array of records to lock
 *
 * Adds current record to aRecords, if it's not already there, and returns aRecords
 */

FUNCTION LockRecords( nSeconds, aRecords )
/*
 * nSeconds                    - Time in seconds to retry a failed record lock
 * aRecords                    - An array of records to lock
 * Array element 1     - Workarea alias
 * Array element 2     - Record number
 *
 * Tries to obtain a lock on all records specified in aRecords. If one record fails,
 * no lock is applied, but tries to lock all again in WAIT_FOR_LOCK_TIME seconds.
 * Times-out after nSeconds.
 * Returns .T. if lock obtained, .F. otherwise
 */

FUNCTION UnlockRecs( aRecords )
/*
 * aRecords                    - An array of records to unlock
 * Array element 1     - Workarea alias
 * Array element 2     - Record number
 *
 * Unlocks the records as specified in aRecords
 */

FUNCTION Ret_Netpath( cDbName )
/*
 * cDbName      - Name of the database
 *
 * Returns the Netpath as specified in IG.INI for dbname. If dbname is omitted,
 * returns the filepath of the first database in the array.
 */

STATIC FUNCTION ShowPercent( mR )
/*
 * mR   - Number of records in database file
 *
 * Shows how far the current record is relatively to mR
 */

FUNCTION        GetSavKeys()
/*
 * Get previously saved keys.
 */

FUNCTION        SetDefKeys()
/*
 * Save current set of keys into aKeys and push it onto stack
 */

STATIC FUNCTION maxpiclen( arr )
/*
 * arr  - array of Pictures
 *
 * Returns the length of the biggest Picture in arr.
 */

STATIC FUNCTION Maxlen( arr )
/*
 * arr  - array of values
 *
 * Returns the maximum length of the values in arr
 */

Static Func to_shadow(t,l,b,r, new_attrib)
/*
 * t,l,b,r              - coordinates of screen
 * new_attrib  - attribute to shadow with
 *
 * adds shadow to screen with dimensions t,l,b,r in the form of new_attrib
 */

FUNCTION StackIsEmpty( aStack )
/*
 * Returns .T. if aStack is empty, .F. otherwise
 */

FUNCTION TimeAsSeconds( cTime )
/*
 * Returns cTime as seconds
 */
```

```
RETURN VAL(cTime) * 3600 + VAL(SUBSTR(cTime, 4)) * 60 +;
         VAL(SUBSTR(cTime, 7))

FUNCTION FReadLn( nHandle, nLines, nLineLength, cDelim )
/*
 * nHandle             - File Handle
 * nLines              - Number of lines to be read
 * nLineLength - Length of the line
 * cDelim              - Delimiter
 *
 * Read nLines lines from a text file with length nLineLength
 *
 */


STATIC FUNCTION Std_Keys( nKey, oTbr )
/*
 * Execute stard keys methods
 */

STATIC FUNCTION ARR_SKIP( n, nPos, nMax )
/*
 * Well, Santo I don't know what you're doing with this routine, but I suppose
 * it works.
 *
 * ???
 */

STATIC FUNCTION Block_Arr( aMenu, i )
/*
 * return aMenu item number i as a block
 */

FUNCTION NET_PATH( Fname )
/*
 * Fname        - File name to return the network path for
 *
 * Returns the Netpath for Fname as specified in IG.INI
 */

STATIC FUNCTION userint(Otype, dbAlias, mode, wait)
/*
 * Otype        - Cache type
 * dbAlias      - db file
 * mode         - not used
 * wait         - seconds to wait
 *
 * User interrupt-routine. Displayed when 2 users simultaneously try to obtain
 * the same lock.
 */

FUNCTION        Rest_win( win_str )
/*
 * win_str      - window to restore
 *
 * restore window with just the window structure returns from save_win().
 */

FUNCTION TimeDiff( cStartTime, cEndTime )
/*
 *  Return the difference between two time strings in the form hh:mm:ss
 */

FUNCTION FileSize( nHandle )
/*
 *  nHandle     - file handle
 *
 *  Returns the size of a binary file
 */

FUNCTION FilePos(nHandle)
/*
 * nHandle      - File handle
 *
 * Reports the current position of the file pointer in a binary file
 */

STATIC FUNCTION         FReadLine( cString, nHandle )
/*
 * cString      -
 * nHandle      - File Handle
 *
 * Reads next line of File Handle into cString
 * Returns:  Logical value: .T. if successful, .F. if EOF()
 */
```

```
FUNCTION        Save_win( t,l,b,r )
/*
 * t,l,b,r      - window coordinates
 *
 * save window with optional coordinates.
 */

FUNCTION        Popup(  Choices         ,;
                                        T                       ,;
                                        L                       ,;
                                        B                       ,;
                                        R                       ,;
                                        Offset          ,;
                                        Header          ,;
                                        lStays  )
/*
 * Choices              - Array of choices
 * t,l,b,r,             - Coordinates for popup menu
 * Offset               - Offset positions from lastpos
 * Header               - Header for menu
 * lStays               - Afterwards, keep displaying?
 *
 * Creates a popup-menu of Choices. Returns the position of the selected choice
 */

FUNCTION        BegMonth(Udate)
/*
 * Returns the date of the beginning of the month of Udate
 */

FUNCTION        EndMonth(Udate)
/*
 * Returns the date of the end of the month of Udate
 */


FUNCTION        TChoice(        nTop, nLeft, nBott, nRight, aItems,   ;
                                aSeleItems, cKeyFunc, nInit, nWin, cColorRef )
/*
 * Menu choice function as described as a standard TBrowse class with the Clipper
 * Libraries.
 */

FUNCTION TrapKeys( nKey, oB )
/*
 *  Standard Clipper function for use with TChoice
 */
```

5.5.3 IGKEY.PRG

```
FUNCTION        IGKey( nSec )
/*
 *      nSec    - Seconds to wait
 *
 * Returns Integer - Clipper compatible INKEY() value, but doesn't take
 * up any processor time. Requires System Manager
 *
 */
```

6 Implementation

This section provides some extra remarks on the source codes, presented in The Intelligen 3-tier system: Source Codes.

6.1.1 Location of the parameter files

We decided to have the parameter files on the back end server instead of on the application host. This because there is a bug in the Advantage or System Manager. It cannot access the Advantage when starting from a local drive. It works under DOS but somehow not under System Manager.

Second the parameter files are only loaded into memory at startup, which does not affect network trafficking that much.

6.1.2 Locking

Although we implemented record locking, we did not use it in the batch posting. This because no one is supposed to access the master file while batch posting anyway, and it would only slow down the performance. So we kept to File Locking.

Record Locking can be useful though if you want to port the online part of the engine

7 Test results

We tested (the performance of) the implemented system as compared to the old centralized host platform. The results can be found in this section.

7.1 Hardware setup

For the performance test we used the following software and hardware setup:

Tier	Component	Configuration
Back end Database Server	Hardware	Pentium 120 Mhz 16 MB RAM 1 GB Harddisk
	Operating System	Novell Netware 4.1 5 Users
	Database Server	Advantage 4.00 2 Users
Application Server	Hardware	Pentium 120 Mhz 32 MB RAM 850 MB Harddisk
	Operating System	System Manager 7.0 3 Users
Backend Network	Cabling	Coax - Thick Ethernet
	Cards	NE2000 compatible - 10 Mbit/s

Table 6: Configuration of test equipment

7.2 Test results

We used a customers application, Agro, to do some batch posting. For the measurements of the response times we took a certain document (SMO) for a certain month (June 1996) and timed how long it took to post it. Table 7 shows the results: Horizontally we see the number of running processes and vertically the configuration. First centralized host and then 3 tiered with a different number of application hosts.

	Number of Processes			
Configuration	**1**	**2**	**3**	**4**
Central Host	8	13	20	35
3T - 1 App Host	11	23	37	55
3T - 2 App Host		12.5	25	27
3T - 3 App Host			14	
3T - 4 App Host				16

Table 7: 3 Tiered vs. Central Host - Response time in seconds

To illustrate the setup we used, Figure 45 shows the hardware setup of a 3 tiered system with 2 application hosts running 4 processes.

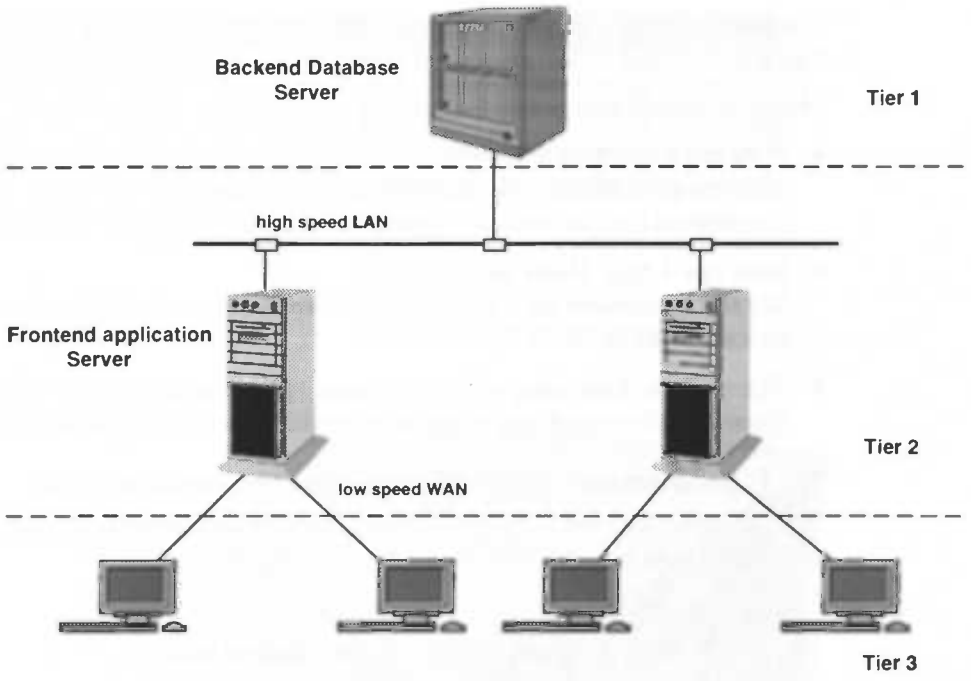


Figure 45: 3 tiered - 2 application hosts, 4 processes

Figure 46 shows these results graphically.

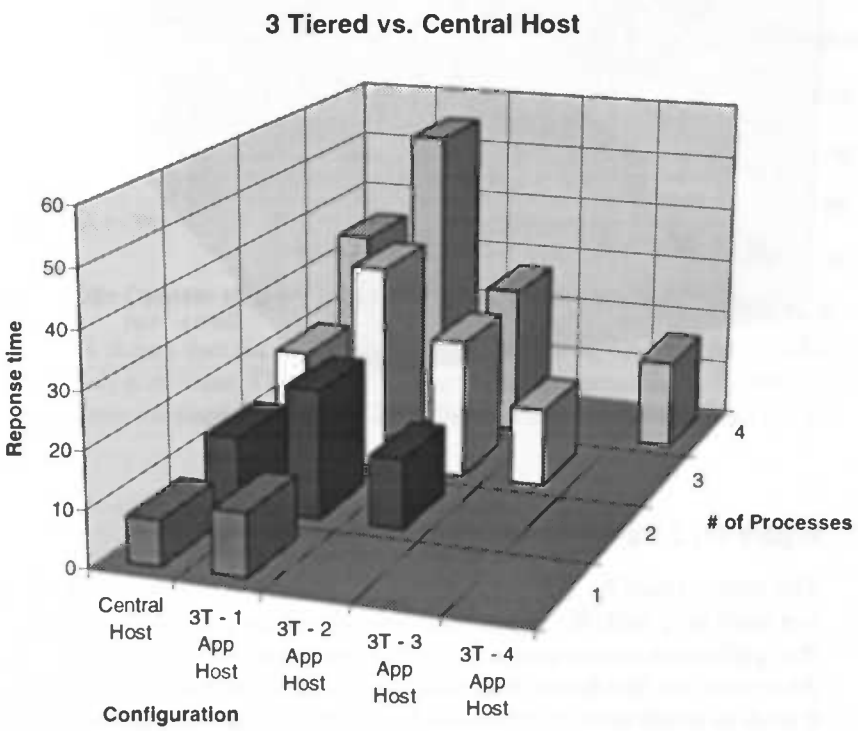


Figure 46: Performance test for 3 Tiered vs. Central Host

The demand time is measured in seconds for posting a certain document for a certain period. Times are in seconds and averaged. We must say that the timings for one process or one process per Application host, are very accurate. When the number of processes increases per Host, the timings could vary.

The number of processes is in this case equivalent to the demand. The response time, logically, increases if the demand increases.

There are some other points to be made.

- How can a test with 4 users be performed with a license for 2 users?
  This was possible because the Advantage treats another user on the same Application host as another "connection" to which there are no limits.
- How can 4 App. Hosts be measured?
  We ran 3 processes on 1 host and timed the fourth one, which only had one process running.
- The test was done using active processes. Of course the number of users can increase with normal use, where most processes are idle while waiting for input.

7.2.1 3 -tiered one application host vs. Centralized host

If we compare 3 tiered - one application host with centralized host we see that the centralized host is faster, which is strange. See Figure 47.

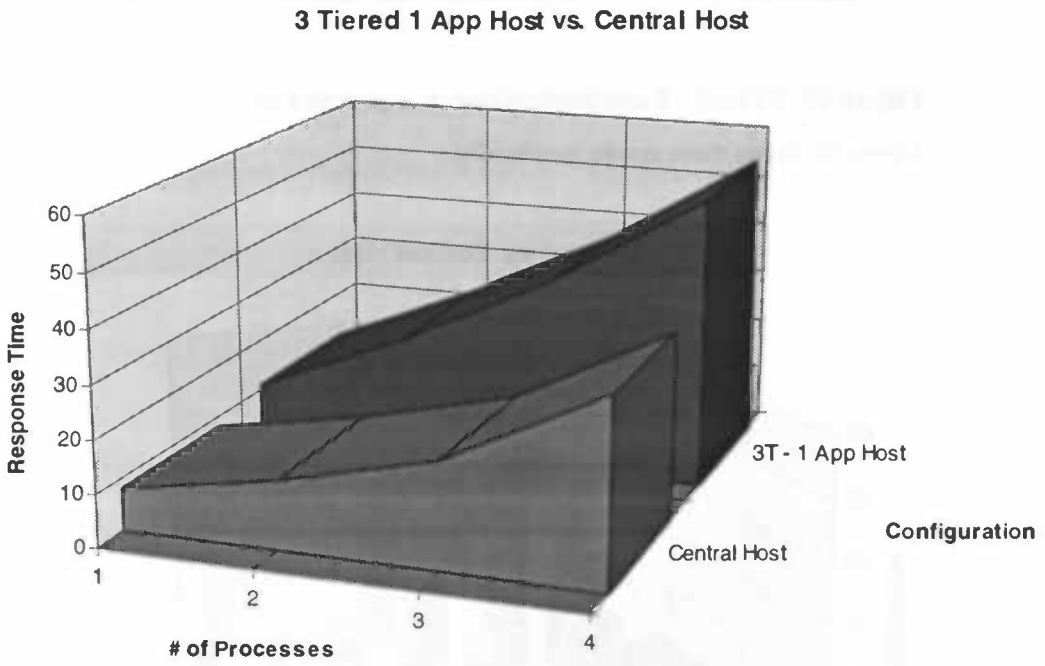


Figure 47: 3 Tiered - 1 App Host vs. Central Host

The reason could be that the combination of System Manager and Advantage does not work very well. We also think some other bugs may be causing this. For instance the application cannot access the Advantage when it is started from a local drive, C:. Also when we first linked it without the IGKEY module, and started up one process, it took so much time of the network card, that others had hardly any time to access it. So that too makes me think that the access of the network drivers by System Manager is not optimal.

7.2.2 Centralized host vs. Client/Server

If we look at the diagonal curve starting from centralized host to 3 tiered - 4 application hosts in Table 7, we get the client/server results. Table 8 isolates these results.

Configuration	Number of Processes			
	1	2	3	4
Client/Server	11	12.5	14	16
Central Host	8	13	20	35

Table 8: Central Host vs. Client/Server - Response time in sec.

Figure 48 shows these results graphically.

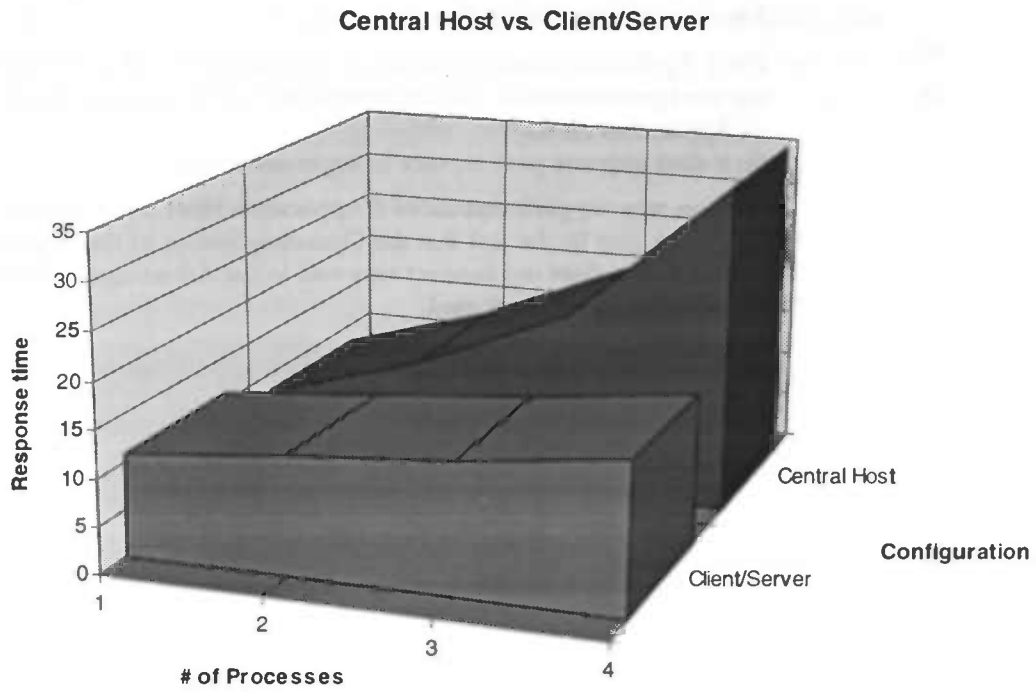


Figure 48: Central Host vs. Client Server

Figure 48 shows that the Client/Server curve is linear, whereas the Central Host curve is definitely not linear. This confirms the theory as presented in Centralized Host Architecture on page nr. 13 and Client/Server Architecture on page nr. 15.

8 Conclusion

Looking at the original thesis:

How can the Intelligen 3-tier Posting Engine be designed and does it improve the Central Host concept in larger networks?

The following conclusion can be made:

- The Intelligen 3-tier Posting Engine can be designed and implemented. The system functions as described in this paper.
- For 1 Application Host performance is not improved. If more Application Hosts are used performance is definitely improved and increases with increasing numbers of Application Hosts.
  So it does improve performance in larger networks.

The reason why the performance of 1 Application Host is poorer than that of a Central Host, may be the fact that the Operating System of the Application Host, System Manager, does not connect very well to the Advantage or that it does not give "fair" access to the network card.

9 Bibliography

"Advantage Database Server CA-Clipper Guide", Extended Systems, 1995 ---- 68

"Advantage Database Server NLM Guide", Extended Systems, 1995 ---- 68

Larry T. Vaughn: "Client/Server System Design & Implementation", McGraw Hill, 1995 ---- 68

Ramez Elmasri/Shamkant B. Navathe: "Fundamentals of Database Systems", second edition, The Benjamin/Cummings Publishing Company, 1994 ---- 68

Santo Chong: "Original IG_ATP code", I&J Software, 1996 ---- 68

10 Appendices

10.1 Appendix A: AXS.CFG Server configuration file

```
;              Advantage Database Server NLM configuration file
;
;   The Advantage NLM reads this configuration file when the NLM is
loaded.
; Values input after the keyword and equal sign are used to
configure the NLM.
; If no value is inserted after a keyword and equal sign, the
default is used.
; Command line values will override the configuration file values.
;
; Number of Connections (-C)
; Default = 5; Range = 1 - No upper limit
CONNECTIONS=15
;
; Number of Workareas (-W)
; Default = 50; Range = 1 - 250 x maximum number of connections
WORKAREAS=50
;
; Number of DBF Files (-D)
; Default = 25; Range = 1 - 250 x maximum number of connections
DBFS=25
;
; Number of Index Files (-I)
; Default = 75; Range = 1 - maximum number of connections x
;    250(max workareas/connection) x 15(max indexes/workarea)
INDEXES=75
;
; Number of Data Locks (-L)
; Default = 75; Range = 1 - No upper limit
LOCKS=250
;
; User Buffer Size (in bytes) (-B)
; Default = 4096; Range = 534 - 65536
USER_BUFFER=4096
;
; NLM Statistics Dump Interval (-M)
; Default = 0; Range = 0 - No upper limit
STAT_DUMP=0
;
; Burst Mode Packets (-P)
; Default = 3; Range = 1 - 16
PACKETS=3
;
; Number of Receive ECBs (-R)
; Default = 15; Range = 3 - No upper limit
RECEIVE_ECBS=15
;
; Number of Send ECBs (-S)
; Default = 3; Range = 3 - No upper limit
SEND_ECBS=3
;
; Number of Worker Threads (-T)
; Default = 3; Range = 1 - 16
THREADS=8
;
; Index Sort Buffer Size (in KBytes) (-Z)
; Default = 8192 KBytes; Range = 1 KByte - 8192 KBytes
SORT_BUFFER=8192
;
; Maximum Size of Error Log (in KBytes)
; Default = 1000 KBytes; Range = 1 Kbyte - No upper limit
ERROR_LOG_MAX=1000
;
; Error Log and Assert Log Path
; Default = root of SYS volume
; The path must be a fully qualified network path,
SERVER\VOLUME:\PATH
ERROR_ASSERT_LOGS=
;
; Semaphore File Path
```

```
; Default = path of first database file opened
; The path must be a fully qualified network path,
SERVER\VOLUME:\PATH
SEMAPHORE=
;
; Transaction Log Files Path
; Default = root of SYS volume
; The path must be a fully qualified network path,
SERVER\VOLUME:\PATH
TPS_LOGS=
;
; Transaction List Elements
; Default = (number of connections + 1) x 9; Range = 0 - 4096
TPS_LIST_ELEMS=
```

10.2 Appendix B: Source code

The codes are based on the original code by Santo Chong[15]

See the source-code section of this paper.

10.3 Appendix C: Link script

10.3.1 atp.lnk

```
#=============================================================================
# Source file: DBFAXS_B.LNK
# Description: Advantage Database Server sample link script for Blinker.
#              Two link script examples are given:
#                 1) Linking in protected or dual mode (Blinker 3.x only).
#                 2) Linking with overlays in real mode.
# Last Update: 11/21/95
# Notice     : Copyright 1995 - Extended Systems, Inc
#-----------------------------------------------------------------------------
#
#
# 1) If linking in protected or dual mode, your main link script should
#    look something like this example 1 (Blinker 3.x only):
#======== Begin example 1: Linking in protected or dual mode ===============
#
# # Pick one of the following:  EXT for protected mode, DUAL for dual mode
BLINKER EXE EXT
# BLINKER EXE DUAL
#
BLINKER EXTMEM LIMIT 2048
BLINKER HOST DPMI ON
BLINKER HOST MESSAGE ON
# STACK 10240

BLINKER PROCEDURE DEPTH 110
BLINKER INCREMENTAL OFF
BLINKER EXECUTABLE NODELETE
BLINKER MESSAGE DUPLICATES
BLINKER EXECUTABLE CLIPPER //F250

FILE ATP, c:\build\OBJ\DBFCDXAX, sminkey, igkey
#
OUT ABP
#
# # Include Advantage Blinker Protected/Dual Mode Communication Library
LIB \BLINKER4\LIB\AXSBCOMM.LIB
#
# # Include Advantage RDD Library
LIB \BLINKER4\LIB\DBFAXS.LIB
#
# # This library should be BLXCLP52.LIB if using CA-Clipper 5.2 and
# #   BLXCLP50.LIB if using CA-Clipper 5.01
SEARCH \blinker4\lib\BLXCLP52.LIB
BEGINAREA
        LIBRARY IG.LIB
ENDAREA
#
# # Clipper library MUST be listed after DBFAXS.LIB
LIB \clipper\lib\CLIPPER, \clipper\lib\EXTEND, \CLIPPER\LIB\DBFCDX
#
#================= End example 1 ============================================
#
#=============================================================================
#=============================================================================
#
# 2) Include this file, DBFAXS_B.LNK, in your main link script as in
```

```
#    the following example if linking with overlays in real mode:
#======= Begin example 2: Linking with overlays in real mode =============
#
# # Only include the following line if using Blinker 3.x in real mode
# BLINKER EXE REAL
#
# FILE MYAPP, MYFUNCS, DBFNTXAX
#
# OUT MYAPP
#
# # Include Advantage Real Mode Communication Library
# LIB AXSCOMM.LIB
#
# # Overlay Advantage RDD Library
# BEGINAREA
#    ALLOCATE DBFAXS.LIB
# ENDAREA
#
# # Include the Blinker link script
# @DBFAXS_B.LNK
#
# # Clipper library MUST be listed after DBFAXS.LIB
# LIB CLIPPER, EXTEND
#
#================= End example 2 ========================================
#
#
#=============================================================================
# These modules should always remain in the root.
#-----------------------------------------------------------------------------
MOD  foxdyn, axdbfdyn, axdbf0, axdbf1, axdbf2, fox0, fox1, axcsupp
MOD  bd0, bd1, bdindexp, bddbfp, bddyn

#=============================================================================
# These modules should remain in the root, but in tight memory
# situations can be overlaid.
#-----------------------------------------------------------------------------
MOD axtag, axutil, axexpr, axmisc, axcnvrt
MOD bdutil, fconvert

#=============================================================================
# Placing these modules in the root will increase performance of index
# creation.
#-----------------------------------------------------------------------------
MOD axxcreat, bdcreate

#=============================================================================
# Placing this module in the root will increase performance when using
# index SCOPEs.
#-----------------------------------------------------------------------------
# MOD axscopec
```

10.4 Appenidx D: Make file

This Makefile assumes that Blinker 4.0 and Clipper 5.2e are installed on the system and are in the file path.

```
Clipper igkey /m/n/a/w
Clipper atp /m/n/a/w
Clipper ig /m/n/a/w
del ig.lib
lib
rem Create new library called "ig " and put in  "ig.obj "
blinker @atp
```

11 Index

Endnotes

1 See: Andrew S. Tanenbaum: "Structured computer architecture", Prentice Hall, 1990

2 See: Larry T. Vaughn: "Client/Server System Design & Implementation", McGraw Hill, 1995

3 See: Larry T. Vaughn: "Client/Server System Design & Implementation", McGraw Hill, 1995

4 See: Larry T. Vaughn: "Client/Server System Design & Implementation", McGraw Hill, 1995

5 See: Larry T. Vaughn: "Client/Server System Design & Implementation", McGraw Hill, 1995

6 See: Larry T. Vaughn: "Client/Server System Design & Implementation", McGraw Hill, 1995

7 See: "Advantage Database Server CA-Clipper Guide", Extended Systems, 1995

8 See: "Advantage Database Server NLM Guide", Extended Systems, 1995

9 See: "Advantage Database Server NLM Guide", Extended Systems, 1995

10 See: Ramez Elmasri/Shamkant B. Navathe: "Fundamentals of Database Systems", second edition, The Benjamin/Cummings Publishing Company, 1994

11 See: "Advantage Database Server NLM Guide", Extended Systems, 1995

12 See: Ramez Elmasri/Shamkant B. Navathe: "Fundamentals of Database Systems", second edition, The Benjamin/Cummings Publishing Company, 1994

13 See: "Advantage Database Server CA-Clipper Guide", Extended Systems, 1995

14 See: TSOM with multiple balances on page nr. 24

15 See: Santo Chong: "Original IG_ATP code", I&J Software, 1996