



The Modern Application Development Framework that lets you 'grow-as-you-go'...

ERROS is a totally new way of creating major computer applications. It is extraordinarily productive and its applications have outstanding performance. Robust, internet ready, scalable ERROS applications are developed

- incrementally, you can 'grow-as-you-go',
- without a physical database design,
- without a detailed system specification,
- and mostly without program creation.

ERROS solves with relative ease a variety of problems for which many developers and even database researchers do not have a generic solution.

ERROS is a total paradigm shift in modern application development methods so cannot easily be compared with traditional development methods. As a result, it cannot be described in a few sentences and any assumptions about how it works might be misleading. The Introduction at the beginning of this manual outlines the main functional features of ERROS and the application areas in which it can be used. This is followed by a detailed description of the unique patented structure of the ERROS Connectionist Database, the advanced functionality of ERROS applications, and the benefits of ERROS for both developers and users.

Rob Dixon, 14th August, 2014

Co-author: **Hein** van der Sanden

ERROS Ltd.,
Boarstall Tower, Boarstall,
Bucks HP18 9UX, United Kingdom
rob.dixon@erros.co.uk



Table of Content

- Introduction.....4
- The Background to ERROS.....6
- The Functional Requirements.....6
- Choosing a Platform.....7
- Why Programming?.....7
- The ERROS database.....8
 - Horizontal or Vertical Record Structure?.....8
 - Entity types and Attributes.....9
- The ERROS Patented Key Field.....9
 - ERROS Key Field Part 1 – Record Context.....9
 - ERROS Key Field Part 2 – Attribute Iteration Identifier.....10
- The ERROS Record Structure.....11
 - From Many Different Files to a Single Indexed Table.....11
 - The Universal Data Type.....12
- Database Handler.....12
 - Exit Programs.....12
- Connections and Relationships.....12
- Hierarchical Data.....13
- Object Oriented.....13
- Integrated Database and Applications.....14
 - ERROS – Incremental Development.....14
 - Applications Defined in Database.....14
- The ERROS Program Structure.....15
- ERROS Semantic Network Diagrams & Relationship Definitions.....16
- Designing a Traditional Application.....17
 - Structured Application Development Methods.....17
- The Principles of ERROS Application Creation.....17
 - Public and Private Data.....18
 - Relational, Network and Hierarchical Structures.....18
- Building the ERROS Business Model.....18
- Menus.....19
 - Functional and Environment Menus.....20
 - Flexibility via menus.....20
- Building the ERROS Application.....20
 - Procedures.....21
 - Views of Data.....21
- Testing.....22
 - In a traditional development environment.....22
 - Testing in ERROS.....22
 - Prototyping.....22
- Network Nodes.....23
 - Multi-dimensional Navigation.....23

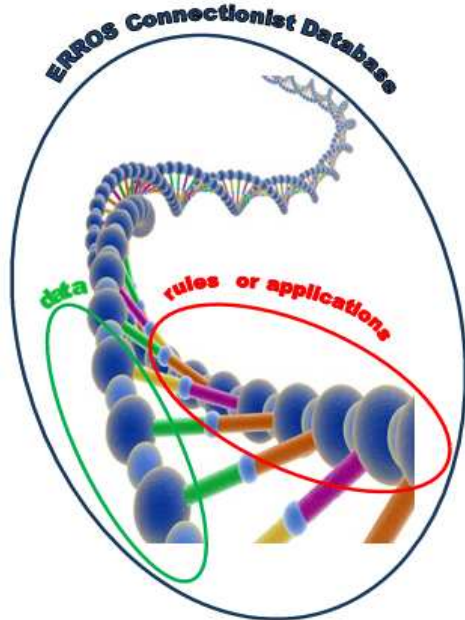


| | |
|---|----|
| Cataloguing..... | 23 |
| Security..... | 23 |
| The Connectionist Database..... | 24 |
| The ERROS Standard Operating Interface (SOI)..... | 25 |
| The ERROS Output Description Language (ODL)..... | 25 |
| Publishing on the Web..... | 26 |
| ERROS screens..... | 26 |
| ERROS Application Selection..... | 26 |
| Product Inventory Screen..... | 27 |
| Order Detail Screen..... | 29 |
| Smart phone screen..... | 31 |
| STIPPLE screens (The ERROS application for the print collection)..... | 32 |
| Initial menu..... | 32 |
| Boarstall Tower..... | 32 |
| A Single ERROS Print..... | 33 |
| A Set of Prints in STIPPLE..... | 34 |
| Multiple Record Identifiers..... | 35 |
| Finding Data in Google and ERROS Applications..... | 35 |
| Archiving..... | 36 |
| Delete states..... | 36 |
| Data Integrity..... | 36 |
| Journalling and commitment control..... | 36 |
| Backup and Recovery..... | 37 |
| Referential Integrity..... | 37 |
| ERROS High Availability Option..... | 37 |
| Transaction Processing & Concurrency Control..... | 38 |
| Performance of ERROS Applications..... | 38 |
| ERROS Today..... | 39 |
| ERROS videos on YouTube..... | 40 |



Introduction

With **traditional application development** methods, data are put in tables and the rules that govern that data are coded in programs. These separate worlds of **data and programs have to be kept in synchronisation**, at best a fragile process. It is this separation of the data and the rules that is responsible for most of the ills of the computer application industry.



ERROS is the **first computer environment** in which the database definitions, the application definitions and the user data are designed to be **totally integrated**, stored in the unique ERROS Connectionist Database. This might be compared to the structure of the DNA string in which both sides are related but independent and are interconnected to form one whole.

ERROS runs under the IBM i operating system on the IBM Power Systems platform and on its predecessors. The single level storage facility of the IBM i operating system, together with its integrated storage management system and its integrated DB/2 database allow any object to be referenced, stored and retrieved by name, without any regard to its physical location. An IBM i application developer simply gives a new table a name. He does not need to specify the actual location and physical attributes of the table and cannot do so.

ERROS, which uses DB/2 as its underlying database, takes this concept a large step further. Rather than using the **"horizontal" record structure of relational databases**, in which related data for, say, a person, are typically stored in a single record, **ERROS uses** its own **patented "vertical" record structure** to allow totally mixed record types and variable amounts of data to be stored in the database without Null values or wasted space.

As a result, ERROS allows entity types, individual entities, attribute definitions and individual attribute values or iterations to be referenced, stored and retrieved by name, and/or number and/or date (and time), without any regard to their physical location. Therefore, an **ERROS application developer does not need to design a physical database schema** and indeed cannot do so.

All entity types and their attributes are defined in the **ERROS Connectionist Database**, rather than in program code, as are application definitions, so no program changes are required when new entity types or attributes are defined. The process of defining the business model and the applications is **totally object oriented**. Applications and data are interpreted at very high speed by the ERROS Database Handler. There are no compiled applications.

Most attributes are stored as connections or relationships. **All relationships are bi-directional** and users can browse all connections in either direction at the same very high speed, allowing them **to find connections that they did not know existed**. ERROS applications turn raw data into a semantic knowledge base. The concepts of ERROS have been **patented**.

ERROS solves, with relative ease, a variety of problems for which many developers and even database researchers do not have a generic solution, such as **incremental development, many-to-many bi-directional relationships, data provenance or public and private data**.



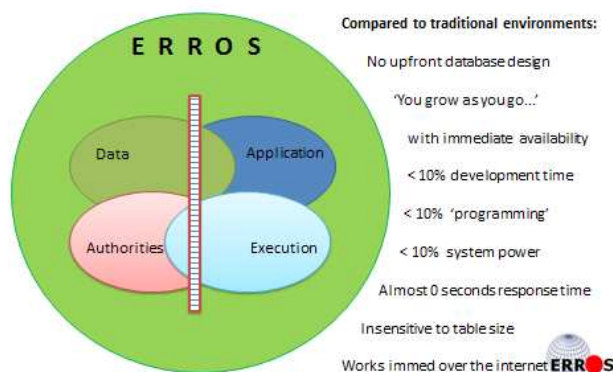
ERROS applications achieve a consistent **almost 0 second response time**, even for most queries, as ERROS does not use joins or SQL. Response times do not noticeably degrade as data volumes grow.

The ERROS database handler communicates with the user, using a **ready to go graphical interface**. This generates HTML and JavaScript on the fly, so all applications **work immediately over the internet** on Pcs, Macs, tablets and smart phones. Developers do not require HTML skills. It also generates PCL5 for printing.

ERROS allows the **incremental creation of advanced applications...**

- that integrate very complex, relational and network data structures with limitless levels of hierarchy
- that can handle totally variable length data without Null values or wasted space
- with navigable, bi-directional, many-to-many relationships by default
- with the ability to handle "Big Data" in its 224 bit address space
- without a detailed up-front specification
- without physical database design or database normalisation
- mostly without any program coding
- with very high security that can be set at the field level
- that can 'grow-as-you-go' incrementally
- with almost 0 second response times, even for most queries, as ERROS does not use joins or SQL
- with scalability – response times do not noticeably deteriorate as data volumes grow
- that can be run in main memory for maximum performance for very high usage
- with an easy to use graphical interface for browsers and a totally consistent method of operation for all applications
- that can retrieve people, things, etc. by multiple identifiers if their names change
- with concurrency control, high resilience, high data integrity and a high availability option
- with much reduced Total Cost of Ownership for development, maintenance and operations

The Connectionist Framework



Data and application definitions are created using ERROS's Open Application Architecture (OAA) using object oriented techniques that allow re-use of all data definitions. These and the application definitions, together with the menus, procedures and the security definitions, are all stored together with the user data in the ERROS Connectionist database. We call ERROS **The Connectionist Framework** as it provides a foundation on which you can build your solutions without needing to spend time building separate layers of database and applications.

Application creation is an extraordinarily productive process in ERROS and **maintenance is dramatically simplified**. Applications can evolve in line with the ever changing real world, solving the costly issue of growing maintenance backlogs. With ERROS, you 'grow-as-you-go'.



ERROS applications are equally suitable for..

- for commerce, including complex transaction processing
- for massive, perhaps global, co-operative databases with different parts of the database being controlled by different businesses or institutions or their experts, scholars or curators
- social business; knowledge sharing & collaboration
- for the apparently unstructured data of the humanities
- for cataloguing collections of any object type, both from the fine arts and elsewhere, and including Union Catalogues containing the holdings of multiple institutions and collectors
- for recording history of any kind, such as employment history or the British 17th century Civil War
- for any domain or database on any topic.

The Background to ERROS

In late 1980, a friend and I needed a computer system for recording history in general, but with particular emphasis on fine art and on cataloguing separately our large collections (50,000 and 40,000) of 17th - 19th century engravings as **90,000 uncatalogued prints are just a pile of paper**. Historical data comes in droplets and then suddenly in a flood. You never know how much data and how many different data types you are going to find so you must be able to have **variable length records** as in 1980, disk space was very expensive. There was no database that could do what we wanted. I was lucky that I hadn't heard of relational databases. If I had, and had studied them (as I did later), I would have found that they could not begin to do what we wanted and might have assumed that this was not possible.



Although I had plenty of IT experience, I was not a trained programmer, having only been on a short programming course in 1965 when I joined IBM as a mainframe salesman. Before IBM, I had been an investment banker and then a stockbroker. After IBM, I was CEO of a computer bureau, and then finance director (and subsequently CEO) of a multinational group of 45 companies, involved in a great variety of businesses. I then became a dealer in and collector of prints. This wide ranging experience, together with my knowledge of history and of the very considerable complexities of print cataloguing, convinced me that the data structure that we required was much more complex than anything I had observed in business or in banking (I had designed a major on-line banking system in 1968 when at IBM).

The only option seemed to be to try and **build a new system myself** (the word database hardly existed in 1980). Clearly, the risk of failure was high, but, if you don't try, ... It was not possible to define the scope of the system up front as, when recording history, you are always coming across new facts, myths and problems. In addition, there is a great variety of art object types, each with its own unique and unpredictable characteristics. Therefore, we needed to be able to develop the system incrementally. But, how to do this?



The Functional Requirements

I felt that the only approach was to build an application development system that allowed **incremental development** and then to use that to build the history system. At least I had the advantage of a clean sheet of paper and was not prejudiced by any database development history as I was unaware of it.

The more important user requirements for the history system, which had to be able to cope with the complexity of the multi-dimensional real world, were -

- The storage of totally **variable amounts of data** for all records, without limits and without wasted space.
- It had to be possible to store **many-to-many relationships** between any records in any entity type.
- All relationships had to be **bi-directional** so that users could **navigate** the relationships and browse through the data in any direction at the same speed.
- It would also be necessary **to store** date dependent relationships, vital for **chronologies** in a history application and much else.
- It must be possible to store **unlimited levels of hierarchy**, in which lower level terms could belong to multiple level higher level terms at different levels.
- For performance reasons, it had to be possible to **find the hierarchies** to which a record belonged **without searching any of the hierarchies**.
- Because names change, it needed to have **a synonym facility** so that any person or "thing" could be accessed using multiple identifiers.
- The system had to be able to handle **vast quantities of data**.
- The system had to be **scalable**, with very rapid response times that did not noticeably deteriorate as data volumes expanded.
- As an upfront design was impossible, the system should be able to **'grow-as-you-go'**.

Choosing a Platform

We then had to choose hardware and operating system. Having considered all other options, I read the Introduction Manual of the recently announced IBM System/38 and was excited by its innovative architecture which included single level storage and virtual memory, 48 bit addressing (unique in 1980), its own integrated database which allowed files to be spread across multiple disks, inbuilt security and communications and its own operating system, CPF, that worked with objects. So we ordered a System/38, a decision that strongly influenced how the system was built. We upgraded to the AS/400 and again to a newer version in 1995 when the AS/400 processor was changed from CISC to RISC with 64 bit addressing (way ahead of Intel and AMD). We have upgraded again since then.

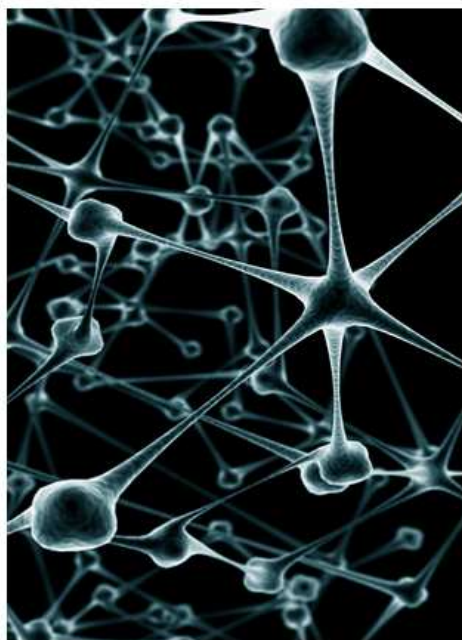


IBM i

Why Programming?

When CEO of the computer bureau, I was of course responsible for all development work and took considerable interest in the details of the systems that we developed to ensure that they would work. Although they worked well, I did feel that the process of **programming** was extraordinarily **inefficient** and costly and that there **ought to be a better way** of building systems. However, I had no idea what this might be and did not have the time then to consider how this might be achieved.





The requirements of the new system were certainly challenging. It occurred to me that **when human beings learn**, as they do all day long, no one specifies changes to the structure of their brains, shuts them down, changes their physical structure or reprograms them, starts them up again, tests the changes and then puts the brains back into production. If this was how our brains worked, we would know nothing.

Yet, this is the way that computer systems are built and maintained and is the reason that they are so difficult to create and change. As a result, they do not deliver what users require within reasonable time and cost scales. The problem is simply the result of **putting the data in a database** and **putting the rules for processing that data in quite separate programs**. This was a fine idea at the beginning of the computer industry, long ago, but is **inappropriate** in today's ever more demanding and volatile business environment.

This **traditional solution** of separating the data from the rules that apply to that data creates its own problem – **keeping the data definitions and programs synchronised**. This separatism produces static rigid solutions whilst the unstructured connectionist real-world problems of their users change dynamically. Static solutions are totally inappropriate for any business that is trying to get ahead or even to stay put.

To my mind, it is this separation of the data and the rules that is responsible for most of the ills of the computer applications industry. Although not a neuroscientist, I feel sure that the brain does not make this artificial separation.

Perhaps there was another way...

The ERROS database

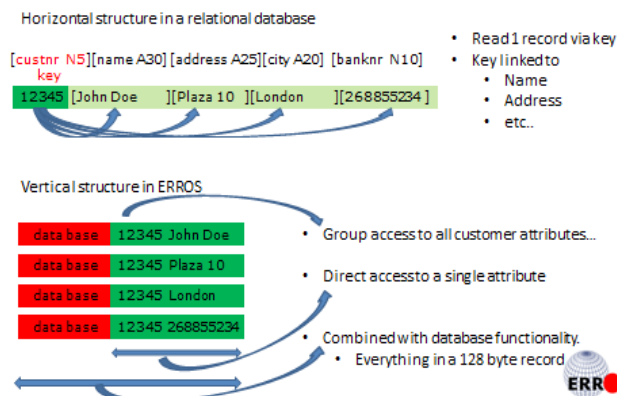
Horizontal or Vertical Record Structure?

The **traditional method** for storing data has tended to be to **put all data** about, say, a person **in a single record**, but, where the amount of data known about each person varies very considerably, a lot of the fields, sometimes a very high percentage, may be empty and much space wasted. This adversely affects performance. There have been database systems that allow variable length records but these have not performed well and those that allow variable length fields tend to have the same problem, as, in both methods, the overflow data is stored in a separate area of disk, which takes time to retrieve.

Because the amount of known data differs so much from any one historical event or art object to another, variable length records are vital for a history or art application. Rather than use a single variable length record with much wasted space and poor performance, the obvious solution seemed to be to use a variable number of **fixed length records**, with each record **storing a single attribute iteration**. With this method, there would be **no effective limit** to the amount of data that could be stored about any individual entity, yet, if the only data known was a Name, then only one record would be required.



Horizontal vs Vertical in ERROS®



The traditional way of storing multiple data fields in a single record might be considered to be a **"horizontal"** structure, whilst storing each attribute in a separate record may be considered as a **"vertical"** structure.

ERROS has some resemblance to column store databases but there are major differences. In ERROS, each attribute is stored as a single record but an attribute can have multiple fields, i.e. multiple columns. In addition, ERROS does not store all values for a single column together in the way that column databases do.

More importantly, in **ERROS**, every single record **has a multi-part key** field so each individual record can be addressed directly, without using a search mechanism. **ERROS does not use SQL**. Although ERROS can store enormous quantities of bi-directional relationships which can be navigated in either direction with the same very rapid response times, it is not a relational database. The ERROS database sequence is described below.

Entity types and Attributes

It seemed that the data would be **best defined** and stored in terms of **entity types and their attributes**. An attribute might contain one or multiple fields.

Entities may need to belong to many entity types with differing attributes, and **inherit** the attributes of the entity type under which they are being considered. Thus a company may be a customer, a prospect, an agent, and a wholesaler at the same time, yet their address only needs to be stored once, accessible for each entity type.

Different entity types might have different length names as record identifiers, so requiring separate files if the name is used as a key field. By using **a patented two stage compression mechanism with hidden internal numbers**, all identifiers would have the same, much shorter length, identifier. Of course, they also need to be accessible by name but, once accessed, this would translate to the internal number.

The ERROS Patented Key Field

The key field of ERROS is the **heart of the database**. It not only provides access to the records but has additional fields that put all data in its context and provide database functionality way beyond the capability of a relational database.

ERROS Key Field Part 1 – Record Context

If the records are given a key field, the first part of which identifies the entity type, say person, and the entity, say John Smith, then a suffix at the end of the key field would allow an enormous number of records about John Smith to be stored yet without wastage of space if little is known about him. As all the records are stored **in key field sequence**, all records relating to a particular entity will be held and retrieved together. If the attribute number is included as part of the key field, after the entity number, then all attribute iteration records relating to one attribute for one entity will be stored together. The entity type number, together with the entity number and the attribute number can be considered as the context of a record. **All data records are stored in this context.**

If there are hundreds of attributes for an entity type, and a user just wants the telephone number for a company, then only telephone number records for that company need to be read – not all the data about it. This is one of many reasons that **performance is outstanding**. If



the telephone number for a company is not known, then no record will be found so null values are not required. By retaining the suffix as part of the key field, multiple attribute iterations can be created, allowing repeating fields. Totally variable amounts of data can be stored about any entity.

ERROS Key Field Part 2 – Attribute Iteration Identifier

At this stage, individual values for a particular attribute for an entity would be stored in the sequence in which they are entered. But, if a user wants to look up a particular employee in his company, this would not be satisfactory if it employed a large number of people. The records need to be stored and retrievable in alphabetic sequence, for people in the sequence of last name and then first name(s). This is solved by using a **two stage compression algorithm** that allows names with multiple words to be **compressed to a number** whilst **still retaining alphabetic sequencing**. The result of the first stage compression, which I have called Attribute iteration identifier in the diagram below, also forms part of the key field, ahead of the suffix. When a particular record is accessed, it will be identified by a hidden internal identifier, the result of the second stage. The suffix can now be used to allow duplicate attribute identifiers if required. If there are duplicates, each will be given a different internal identifier. Thus every single attribute iteration will have a totally unique identifier, and, since this is a key field, **every attribute** iteration can be **accessed by key without a query language**.

The key field design ensures that there can **never be a duplicate key**, however many records there are. This means that John Smith's telephone number can be found without having to read all the other data related to him. No query language is required. This is so different from a relational database where all data relating to a contact is generally contained in one continuous record and, to find any data, SQL or some other mechanism is required.

The space used for the attribute iteration identifier in the key field can also be used to **identify records by date and/or time** or by number (perhaps product number or telephone number) or a code. A sequence type marker in the first part of the key field keeps these different sequences separate and so allows attributes to have up to three indices – say, name, and/or number, and/or date (& time).

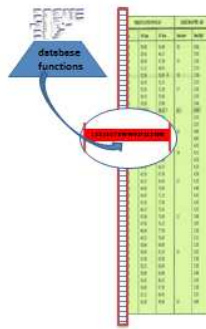
Thus all contact names can be stored alphabetically (last name then first name but displayed first name, last name). Records can be retrieved instantly by name. If the required record does not exist, only one record needs to be read to establish that it isn't there. In a sales order, an order line can be retrieved instantly by either product name or product number, without the need to search all the lines in an order. Chronological lists can be in date sequence (YYYYMMDD but displayed DDDMMYY or DDDMMYYYY or MMDDYY, etc.) or date and time sequence. This **multi-part key field** solves the problem of handling lists as the records can be retrieved in any of these sequences without sorting or using a query language.

Simplified Schematic of Key field for all records

| | | | | |
|--------------------|---------------|------------------|--------------------------------|--------|
| Entity type number | Entity number | Attribute number | Attribute iteration identifier | Suffix |
|--------------------|---------------|------------------|--------------------------------|--------|



ERROS® database key...



- **key of 28 bytes**
 - Binary
 - Double compressed
 - 224 bit address space
 - Allows max 2^{224} records
- **composed of 17 fields**
 - definitions
 - connections
 - navigational
 - Bi-directional
 - Multi-dimensional
 - ...



The actual key field is somewhat more complex and requires 28 bytes, which is a massive 224 bit address space. This is split into 17 subfields. Of course, all the possible space will never be filled, but ERROS was designed to handle 'Big Data' from the beginning. The key field allows each entity type to have 1 billion entities, each with up to 1 billion attributes, each of which could have 1 billion iterations for each entity, solving the problem of repeating attributes. **The key field structure forms part of the ERROS patents.**

As ERROS accesses records by key and does not use direct addressing, the design will run on a 64 bit system. There are occasional rumours that IBM will change their Power Systems architecture to 128 bit addressing. If they do, ERROS will not require any change.

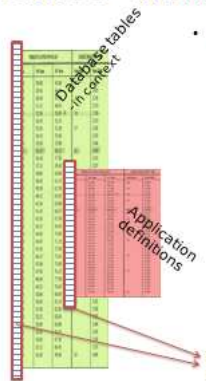
The ERROS Record Structure

The page size of the System/38 was 512 bytes (much larger on AS/400 and successors) so it made sense to divide this into 4 records of 128 bytes each, split into three fields -

~~Key – 28 bytes with~~

- **Key – 28 bytes with** 17 subfields – described above
- Control Data – 15 bytes with 12 subfields – discussed below under the heading 'The Universal Data Type'
- Unique – 85 bytes.

ERROS® database key and record...



- **Records of 128 bytes**
 - Database tables
 - Application definitions
- **Key of 28 bytes**
 - composed 17 fields
- **15 for control data**
 - composed of 12 fields
 - record type/audit trail/security
- **80 for Unique**
 - 65 for data + 20 for connection data -or
 - 80 bytes text

Same keystructure but unique key



The **field "Unique"** can store 65 bytes of user data and 20 bytes of relationship data or 80 bytes of text with 5 bytes spare. Where 65 bytes of user data or 80 bytes of text is not enough, multiple records can be used, without the operator being aware of this.

20 bytes of connection data is adequate to store enough information to identify the key of the related record to allow navigation.

This structure is used for every single record in the connectionist database.

From Many Different Files to a Single Indexed Table

The history application, and many business applications, would require at least a few hundred entity types. As the chosen platform (S/38 – AS/400) had single level storage and files that could occupy multiple disk volumes, it no longer made any sense to have large quantities of separate files, one for each entity type. The key field and record structure described above meant that all records would have the same key field and record length, so that they could all be stored in just one table in the integrated system database, taking advantage of its stability and its high performance binary tree index. For operational reasons, that single table was later split into ten, all with identical layouts with the same key.



The Universal Data Type

A Universal Data Type was necessary to **allow all data of all types to co-exist in any table**. This is achieved with a 'Control Data' field in each record. This includes a 3 character data type code, a transaction number field, and fields to store special security options for each record (attribute iteration). The latter are discussed later under Security. The Universal Data Type can store both text and non-textual data.

Database Handler

With separate files, one or more new programs would have to be created for each new file, whereas putting all the data in just one table means that only one Database Handler is needed to add, retrieve, update and delete all records, whatever the entity or record type. The performance benefits are very significant as is the reduction in development costs. As new entity types are defined, no new code for database handling needs to be created and no new backup procedure is required.

Without separate files for each entity type, there is **not the constant need for closing and opening of files** when users navigate relationships, with further very considerable performance benefits. **No slow performing joins or SQL are required.**

It seemed sensible at the beginning of development to write separate code for the interface which could then be used for all applications. This meant yet another large improvement in performance. As the same main programs are used by all users, whatever their application, and, whatever the number of users, only one copy of each program needs to be in main memory. These main programs are used in conjunction with the exit programs referred to below.

Exit Programs

Except for a few control data types, such as menu records, the database handler does not need to be aware of the **different types of record**. Based on the 3 character code, it will call an "exit" program that relates to the record type code in the control data. The same 3 character code may be used for many different attributes. **One exit program can handle many record types**, so not many will be required. They can be used in multiple applications. These exit programs do not access the main database – they receive data from the database handler, break it down into its constituent fields where necessary, do any calculations required, perhaps display it or pass it back to the main programs for display, or allow users to update the on-screen data (if their security permits) and then pass the changed record back to the database handler which updates the database. Exit programs can **also be used to access external data** and pass it to the main programs, perhaps for display or for loading into the main ERROS database, or they can update external data. Users are unaware of the existence of exit programs.

Exit programs are called dynamically, so changes to them or newly created exit programs do not require any changes to or recompilation of the main programs.

Any exit program can undertake any function that any program could have, such as accessing PC type folders and files or accessing web services. **Users can create their own exit programs.**

Connections and Relationships

Most attributes would store connections or relationships or text. The valid connections for an attribute would be defined in the database, using other connections. John Smith's address(es) would not simply be text typed into an attribute. There would be an entity type "Address" in which all addresses would be stored. When putting up an address for, say, John Smith, this would be a bi-directional relationship with an address record. Once he was linked to his



address, you could navigate to that address in the entity type "Address" and the attribute "address of" would show to whom that address belongs – obviously John Smith but maybe also to members of his family or to a business that he runs from home. You could **put dates on address relationships**, so that you can see who was at an address over time and what addresses any person or business had had, with both lists being in date sequence.

As all attributes can have multiple iterations, by default this applies to both sides of a relationships so **many-to-many relationships are also available by default**.

As **all relationships are bi-directional**, it is always possible to look up a record from the reverse viewpoint. So, to find out the owner of a telephone number, look up the number in the entity type "telephone number" and ERROS will list in alphabetic sequence the people or businesses that use that number.

Hierarchical Data

Another problem was the storing of **limitless hierarchical data**. If you take a family tree, there may be many John Smiths in different branches who were born at totally different times. You have to uniquely identify each one by reference to their parents, grandparents, great grandparents, etc. You may not know their date of birth and many John Smiths from the same family may have lived at the same address at different times. This means that you need a variable length key field as someone born last week will require a longer identifier than someone born 500 years ago. If we assume a gap of 25 years between generations, then 500 years is 20 generations. If any name can be, say, 64 characters long then you need a key field that is 64 X 20 long - 1,280 characters. 5,000 years would be 200 generations - 12,800 characters.

Since databases cannot handle variable length keys, this implies that you would need a separate file for each different key field length, hardly a practical solution. The human brain seems to cope with this issue and, on the assumption that the brain does not have a variable length key field function, there must be another way of doing it. I evolved a solution that allows **a billion levels in a hierarchy within the 28 byte fixed length key field**. The length of the key does not change as you go down the levels.

This also formed part of the ERROS patents. Users can **travel up and down all hierarchies** without limits, and **exit or enter at any level**. They can find out in which hierarchies a record is contained without having to search any of the hierarchies, with obvious performance benefits. Records can belong to multiple hierarchies of any type of record, allowing unlimited classification, categorization, ontology or taxonomy. This also provides a structure for handling groups and sets, functions that most systems find difficult to handle. This single database building block is a vital feature, also used, for instance, for storing relationships with relationships or for creating menus and sub-menus.

Object Oriented

After the initial design of the database, it became clear that one tiny part of the application that would require regular changes could be defined in the database rather than in program code. It wasn't a giant leap to realise that **all** the database definitions could be stored in the database by creating an entity type called "Entity Type" and another called "Attribute". With these, the business model could be defined in the database. These entity types and attributes can be defined in the everyday terminology of users, in any language whose character set can be handled by the keyboard. Other entity types might be "Customer", "Address", "Product" and "Sales Order". Attributes might include "name", "number", "name and number" (two fields in one attribute), "date", "postcode", "orders placed" and "product ordered". **Each attribute is defined just once but can be used many times** simply by connecting it to entity types. For instance, "Customer" and "Product" and many other entity types might have an attribute



"name and number". The whole database was designed from the beginning to be object oriented with entity types and their attributes being defined just once and those definitions used in multiple applications.

Individual entities inherit the attribute definitions for the entity types to which they belong. The internal entity type and attribute numbers are allocated automatically by the system. Since these are part of the key field of each record, adding new entity types or attributes has no impact on existing data definitions or user data. **Physical database design is no longer necessary.**

Integrated Database and Applications

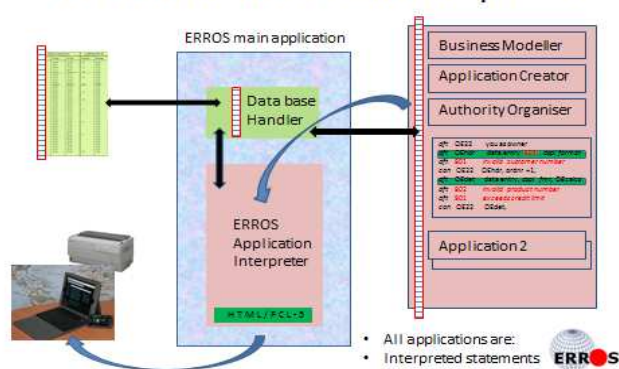
ERROS – Incremental Development

There was still the problem of incremental development. In a traditional environment using relational databases, adding a new field to a file means that every record has to be changed together with all programs that access that file and the system has to be shut down whilst this is done. This new ERROS data model means that, since **the database can be changed whilst it is live**, the database definitions can be changed without any computer programming and so without shutting down the system, a major step towards the goal of incremental development.

Applications Defined in Database

The next step was to realise that, if the total data model was defined in the database, and the data model and the user data that it defined were accessed by a single Database Handler, then individual applications would simply provide a route to the relevant parts of the business model and so to the user data. After creating an entity type "Application", with each application being identified by "name and number", each application would be an entity in this entity type and would be related to the appropriate entity types and, at a lower level, through menus, to the

ERROS® internal: as an interpreter



attributes of each of those entity types that are required in the application. Thus the **application definitions, including menus and procedures, could also be stored in the database** and, except perhaps for a few exit programs, usable in multiple applications, **no programs would need to be coded or compiled** for a new application. The **same main programs**, including the database handler, are used for defining the business model, building the applications, defining security and operating all the applications. **They interpret all the data and application definitions** and the user data at very high speed.

Applications are created and changed by updating the database, not by coding and compiling programs. There is no longer a set of programs for each application and **the concept of the compiled application has gone.**

Since everything is defined in the database, **adding new entity types or attributes has no impact on existing data or applications nor on the database** handling and interface programs. Simply by adding to or changing the data and application definitions stored in the database, the problem of incremental development is solved. I suggested earlier that the separation of the data and the rules in traditional applications was responsible for most of the ills of the computer application industry. In ERROS, they remain totally integrated. The skill

levels required for ERROS development are rather less than those needed for traditional development methods.

The separatism of traditional development methods has been replaced by the connectionism of the real world.

The ERROS Program Structure

As explained above, the ERROS Business Model, the Application definitions and Authorisation definitions are all stored in the ERROS Connectionist Database together with the user data.

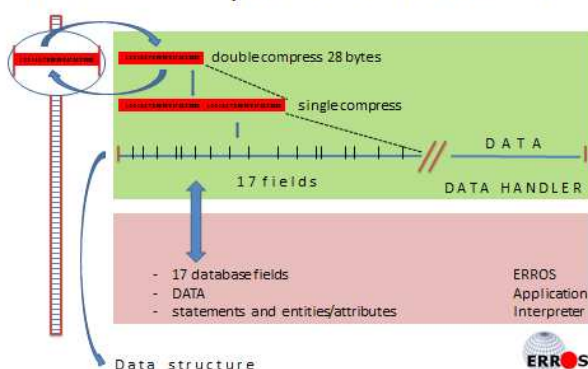
These are all retrieved and updated by the ERROS Database Handler. The only other main programs required are the ERROS Interface Program, which handles HTML and JavaScript generation, and an exit program. There are no separate printing programs as all printing is done by the interface program, with the report layouts defined in the Connectionist Database.

These programs together, perhaps with other exit programs, interpret all the business model and application definitions and execute the applications.

All of ERROS processing is done in one single ERROS Main program group. Once started, ERROS presents a list of the available applications from which you choose one. The main elements of that Main program group are:

- **The ERROS Database Handler.** This calculates the key field, retrieves record(s) and passes them to the
- **ERROS Application Interpreter,** which executes all ERROS Applications. It interprets records received from the database handler. It may request more records from the DBB Handler or it may pass them to
- **The ERROS Interface Program.** This generates, on the fly, the HTML and JavaScript necessary to display the records using a browser or generates PCL 5 for printing.

The ERROS® key and the Data Handler..

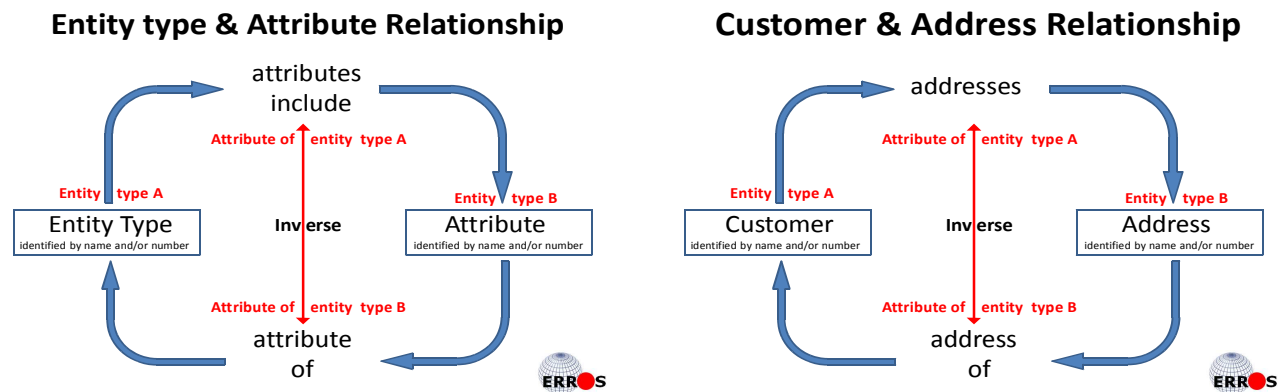


The data definition tables and the application definition tables have the same key structure and together they form the ERROS Database. As explained before, the ERROS key is built from 17 fields that identify the content of the record and contains all database functionality. This picture shows how the Interpreter passes these 17 fields and the record content to the Database Handler, that double compresses these fields into a unique 28 byte binary key and writes the record to one of the ERROS tables.

When a single record is to be read, the DB Handler defines the key, reads the record, extracts the data and passes that to the Interpreter. When a group of records must be read, such as an application, or a complete order, etc. part of the 17 fields key defines the group and the records of that group are read and passed to the interpreter.



ERROS Semantic Network Diagrams & Relationship Definitions



The two semantic network diagrams shown above illustrate the structure of relationship definitions. The **first diagram** shows the relationship between **Entity Types and Attributes**, used in the definition of the ERROS database, illustrating how ERROS is defined by itself. The entity type "Entity Type" has an attribute "attributes include" which is a relationship with the entity type "Attribute", with the inverse relationship being stored in the attribute "attribute of". This defines how any entity type can be connected to any attribute and so inherit the properties of that attribute, without the need to redefine it. Since ERROS allows many-to-many relationships, entity types can have multiple attributes and attributes can be attributes of many entity types.

The **second diagram** defines user data and shows the relationship between **customers and their addresses**. The entity type "Customer" has an attribute "addresses" which is a relationship with the entity type "Address", with the inverse relationship being stored in the attribute "address of". This defines how any customer can be related to any address. With many-to-many relationships, customers can have multiple addresses and there can be many customers at the same address.

When developing ERROS applications, there is **no need to draw these diagrams** – they are included here to assist understanding of the simplicity and consistency of the structure of ERROS. No developer can understand the complete structure of a complex database as a whole. With ERROS, he or she only needs to understand the separate parts, as **ERROS automatically integrates** all these into the working whole. Thus, if the developer is sure that every part of the structure of his database has been correctly defined then he can be sure that the whole is correct, even though it may be far too complex for him to understand as a whole at an instant of time

These diagrams show that there is no difference between the definitions of the structure of the ERROS database and the definitions of the structure of user data. The same mechanism defines them both.

It only takes four simple steps to define a relationship in the database and this is done without any special syntax. No special syntax is required for system development. Developers use the everyday terminology of the business, in any language that can be handled by the keyboard.

Relational databases and SQL use special syntax such as 'triples'. A book has an author. In SQL, this might be defined as 'hasAuthor'. In ERROS you can use spaces in names and could say 'has author' which is much more user friendly.



Designing a Traditional Application

In traditional application creation methods, the first step is normally to **create an application specification**. This is a complex document that can take some time to create, and has to be understood by both the users and the developers. The user(s) have to **sign off the specification** when the document has been completed. Whilst the users want a result – a working application – from this process, they may not fully understand the specification, in part perhaps because they still have to complete their everyday tasks and do not have the time to really focus on the detail. Some users even resent the time that is taken up by creating the specification. All they want is easy to use software that does the job, delivered as quickly as possible, at a sensible cost. At the same time, they require a system that is stable and always available, without breaking down. Speedy delivery often fails to deliver stable systems.

The traditional detailed up front specification is required because of the separation between database, applications and interfaces and traditional development methods have to keep these worlds in synchronisation.

The common method of developing traditional applications is to design a database and create program code with the required functionality.

Structured Application Development Methods

Structured Application Development methods were developed such as Waterfall, later followed by Agile methods such as the more modern Scrum. All are somewhat sequential methods and they do not guarantee on time or on budget projects or satisfied users. The bigger the project, the more likely that changes will be required, with inevitable time and cost overruns, particularly if the changes are at the end of the development cycle. Many development projects have to be reworked because they fail to meet user's needs and some are simply cancelled.

The Principles of ERROS Application Creation

In ERROS, business modelling and application creation takes place **without any programming**, unless a new exit program is required.

It only takes four simple steps to create an ERROS application:

1. **Extend the ERROS Business Model** using the ERROS Business Modeller. No detailed physical file design or schema or normalisation of data is required. No special syntax is required – developers use the standard terminology and language of the business that their users already know. No new files are created to store the user data as all data definitions and user data are stored in the existing files in the ERROS Connectionist Database.
2. **Create an ERROS Application** using the ERROS Application Creator. Whereas traditional applications are defined in programs, ERROS Applications consist of menus and procedures, built over the business model, and defined as application definitions in the Connectionist Database. They act as filters and control access to those parts of the business model and user data (also stored in the Connectionist Database) appropriate for the required function or process (e.g. sales order entry). ERROS applications also define the level of security required for the actions (e.g. read, add, update, delete, copy, print, etc.) that can be performed by each operator on each part of the user data available in the application.
3. **Authorise the new application** using the ERROS Authority Organiser. This authorisation, which allows very tight control of security, down to the attribute level, is also stored in the ERROS Connectionist Database.



4. **Test the finished application.** Testing is also done in every stage of the incremental development of the application.

All changes to the business model, to application definitions and to the user data are recorded in the ERROS Audit Trail, providing a level of documentation rarely achieved with traditional system development.

Public and Private Data

Since all records in the database have an identical layout and key field, another element – user number – was added. This had been allowed for in the 28 byte key field, so allowing the concept of public and private data. **This applies to data and application definitions as well as user data.** Thus there may be public and private entity types, public and private attribute definitions and public and private applications. Users can add their own privately defined attributes to publicly defined entity types but these will not be visible to other users. Users can create private applications that use both publicly and privately defined entity types and the publicly defined entity types can have their own privately defined attributes mixed with those already defined public ones that they require. Thus data viewed as a single logical record from a user's view point may contain both public and private attribute values. Operators from different companies or institutions would see the same data in the public attributes but different data in the private attributes.

User menus can be created within a publicly defined application. These can change the way the data is processed and/or the way it is presented both on screen and on printed reports, as all output definitions are also stored in the database. Once again this would not affect any other users in any way.

This flexibility makes it easy to **create a standard application that can be adapted to suit the needs of individual users without changing the original application**, ideal for creating an ERP or any other package.

Relational, Network and Hierarchical Structures

Relational, network and hierarchical structures are all supported in one totally integrated ERROS database. One consistent simple mechanism for defining bi-directional relationships or connections is used throughout ERROS to define entity types, attributes, applications and their menus and procedures and also all relationships and hierarchies in user data, all stored **in the same integrated database**. Conceptually, this is a single multi-dimensional semantic network that integrates all definitions and user data, yet, because there can be both public and user versions of both definitions and user data, each user group, company or institution may see a slightly or dramatically different version of that semantic network. No one will see all versions.

Building the ERROS Business Model

As stated above, traditional development methods require an up-front detailed physical database design, before coding can begin. This can only be done when all the data to be stored has been defined. This is not necessary with ERROS as no detailed up front specification is required and there is no need to design a database schema. **The whole process is done incrementally**, and starts with building the business model.

A variety of entity types and attributes are already defined in the ERROS Open Enterprise Business Model. Developers can use these where appropriate and extend them. They can **create new entity types and new attributes**. They can **add new attributes** to existing ERROS defined entity types and they can use existing ERROS defined attributes and those that they have newly defined with their new entity types. The attributes are added to the entity types simply by **creating connections**.



This is an object oriented process that can be carried out **by multiple developers working concurrently**. Those attributes that are to store relationships, perhaps the majority, need those connections to be defined, along the lines of the semantic network diagrams shown above. The developer selects an entity type and one of its attributes, then -

1. relates that to the appropriate entity type,
2. selects the attribute of that entity type in which the inverse relationship is to be stored,
3. selects the entity type in which that relationship is to be stored, usually the same as the first
4. and then selects the identifying attribute of that entity type.

This whole process only takes perhaps 30 seconds for each relationship and requires no special syntax.

Building the ERROS Business Model will be undertaken after the developer has received answers from users to the question - "What types of 'things' (e.g. customers, products, orders, employees, etc.) do you want to store data about?" in order to identify the entity types, followed by the question "What information (e.g. name, number, address, products ordered, price, etc.) do you want to store about each of those?" to identify the attributes required. For each attribute, the developers will need to establish whether they are relationships and, if so, define those. For existing ERROS defined attributes, the relationships may have already been defined, but the developer can override these definitions where necessary.

Because **the business model is open ended**, additional entity types and attributes can be **added at any point** if they have been missed, even after the application has been created. **Relationship definitions can be changed at any time.**

The developer does not have to create any new files nor does he or she need to be concerned with how or where the data will be stored as ERROS handles this automatically.

The next stage of the ERROS development process is to define the application, but, before that process is explained, some understanding of ERROS menus is required as they are the basic building block of ERROS applications.

Menus

As in many systems, after any action, a menu will be presented, from which the user can make a choice in order to proceed. These menus are stored in menu records defining which data or menu or procedure is to be retrieved next. The options available at each decision point or node are determined by **application specific contextual menus** stored in the Connectionist Database. If there are no menu records, then there is no lower level data available and none can be added in that application. Menu records control whether the next records to be read after the menu record are further data records or other menu records. Menu records also **control a variety of security, processing and display options** about the data and menu records to which they lead. **Menus are generally application dependent** so that the menu at any context can be different from application to application. There can be user based menus that override the standard menus.

Higher level security can be put on individual menu records in a menu to limit access to those menu items to more senior operators.

In transaction processing, entries in menu records determine when **commitment control should start and end**, and menu records that are part of the transaction definition determine which records are to be added, changed or deleted as part of a transaction.



Menus can contain any number of menu records and menus can be nested – each menu record can lead to another lower level menu, each lower level menu record can lead to another and so on. Of course, menu records can alternatively lead to data records and selection of one of these may then lead to further menu records. Where there is only one record in a menu, ERROS would normally read that and move to the next data or menu record without stopping.

Menus can only be changed in an Application Creation application.

In an ERROS Application Creation application, the menus that relate to each application are user data rather than meta data, but as soon as a user operates an application, that user data becomes meta data and the user of an application is not able to change the application definitions and menus whilst he is operating that application. Most users would not have authority to business modeller or application creation applications.

Functional and Environment Menus

As explained earlier, when a data record has been read, ERROS will always look for a default menu record that relates to the context of the data record. However, there can also be functional menus – menus that are selected depending on the function applied to a data record. Thus if a new record is added, it is possible to define that ERROS must subsequently access an "add" menu, probably application dependent, so that, for instance, when adding an order line on a sales order, an ERROS procedure, invisible to the operator, can update the order header, allocate stock, etc., as part of a transaction. There can also be "change", "delete" and other menus. An "inquiry" menu can be used to control the types of query that can be made, etc. A printing menu can define the layout of printed reports. These functional menus can be used to undertake tasks that in a traditional system might be done by trigger programs whereas in ERROS there are no trigger programs and no new programming is generally required. If a functional menu has not been defined, a default menu will be used.

There is also a facility for using an HTML environment menu for use when ERROS is being accessed by a browser, so that, for instance, images can be displayed over a browser but image URLs are not sent to a 5250 display. This does not mean that a complete duplicate set of menus is required for the two environments.

Functional, environment and default menus can all be used with each other.

Flexibility via menus

A menu record is always accessed after a data record has been processed. The ability of ERROS to have a great variety of menus at any point, including the functional and environment menus described above, allows extraordinary flexibility in the way that data can be accessed, processed and presented, both on screen and in printed output.

Building the ERROS Application

The data model has already defined using the ERROS Business Modeller.

Application development can begin long before the business model has been completed. As no new interface has to be created, developers can **very quickly demonstrate their progress to users** and get immediate user feedback on whether the developers have properly understood what the users wanted and also whether the users had actually requested what they really wanted. If changes are requested by the users, the main programs do not need to be changed – only modifications to the business model and/or application definitions are required.

A major **concern with traditional application development** is in ensuring that the new application performs with **really rapid response times** that do not noticeably degrade if data volumes dramatically increase. The performance depends on precisely how the application and



its database are designed, and can vary dramatically between separate applications that are in reality not that dissimilar. The performance depends too much on the skills of the developers.

In ERROS, no attention is given to achieving rapid, consistent response times as these hardly vary for most applications. This is ensured by the **consistent, key driven direct access to each record or group of records** in the ERROS database.

The **first step** in creating an ERROS application, after naming the application in the entity type ERROS Application, is to **relate that application to the entity types** required in the application and then to create, for each entity type, a first level menu that contains a menu record with the name of the identifying attribute – e.g. name .

The **second step** is to **connect those entity types to their attributes** that are required in the application. Some attributes may be at a lower level, dependent upon higher level attributes – for instance, the attributes “department of employee” and “job function of employee” would be at a lower level below the attribute “employees” as they are dependent on the person being an employee of the company.

Procedures

Where there is more than one menu record in a menu, the previously read menu record may determine that the records should be displayed to the user so that he can select one or it may determine that the records are to be read one at a time, as in a procedure, going on to the next data or menu record and so on until there are no more records to be found. In such a procedure, ERROS might then go backwards, looking for further records and moving forwards and then backwards until it returns to the starting menu. It will then read the next record in that and move forward again and then backwards, only stopping when it has finished processing all the records in the starting menu. This might be used for transaction processing or for creating complex reports. There can be tests in a procedure, that, if valid, instruct ERROS to exit the procedure at other points. Menus in a procedure will not generally be visible to a user.

Procedures are constructed with nested menus using the hierarchical functions of ERROS. They can become very complex with a very large number of levels and tests at each level.

ERROS procedures can be used in a great variety of ways. For instance, when a customer name is selected from a list, it might be displayed with address, telephone number(s), contact name and date and value of last 3 orders but omitting any other attributes. A procedure will retrieve the necessary data, which is stored in multiple records and display (or print) them as a single logical record. Alternatively, when **entering sales orders**, ERROS can look for the best price for the customer, searching first for prices or discounts for a product or product group for that particular customer, or, if those are not found, perhaps for discounts for the geographical area of the customer, etc. If no special price or discount is found, then ERROS would retrieve and display the standard price. Prices can be date and time related and have an end date and time, so that a special price for a limited time can be entered in advance. ERROS will automatically apply this when appropriate and then revert to the standard price. This can all be achieved by selecting the various options in the development process and without any programming.

Views of Data

As already described, ERROS stores the data for each attribute iteration in separate physical records, and a separate menu record is read for each attribute before ERROS accesses the data in the attribute. The menu records define how the data in that attribute is to be displayed. For instance, in order entry, all the detail of an order entry line might be displayed, but, in some other application, perhaps only the product name, product number and quantity ordered might be required. These options are easily defined in the appropriate menu record.



Testing

In a traditional development environment

There are many objectives of testing in a traditional development environment. A **formal test plan** has to be developed. Its aims can include –

- is the program code error free?
- does the new application fulfil the objectives of the system specification?
- is the new application or functionality correctly integrated and compatible with existing systems?
- are response times sufficiently rapid?
- are response times consistent?
- load testing & scalability - will the system continue to work under heavy usage and with large data volumes?
- will the system be accepted by the operations department?
- does the installation process work correctly?
- does backup and recovery work?
- does the new application and data have proper security?
- is the new software adequately documented?
- Even if the system provides the specified functionality, does it really meet the needs of the users?
- Is the method of operation acceptable to users?

Testing in ERROS

In ERROS, application testing is very much simpler as **any change at any moment can be tested as you 'grow-as-you-go'**, so testing can be completed much more quickly. Although the application must be thoroughly tested, no testing is required for the main ERROS code which includes the database handler and the interface program. If any new program code has been created, it will only be in exit programs. These generally have a fairly simple structure, and should be simple to debug if necessary.

As users can be shown work in progress, they should be able to see at an early stage in the development process if this does not meet their needs, so that corrective action can be taken if necessary.

Because of the way they are constructed, **ERROS applications** should always have outstanding response times, and **should not need stress testing**.

In terms of tables, programs and other objects, there is nothing to install unless this is the first ERROS application on the system, nor are new backup procedures required as ERROS does not create new tables, only adding to existing ones.

As all changes to data and application definitions are stored in the ERROS database and all changes are always journalled and also recorded in the audit trail, the documentation of all **changes to all definitions is always to a consistent high standard**.

Prototyping

Because data definition and application development are incremental processes in ERROS and no detailed up front specification is required, ERROS can be used for prototyping application development. Users can be shown the planned and working solution to their problems in stages, before they have forgotten why they requested some of the features that they thought were so vital. Users can see at all stages whether a new system is going to meet their needs. It can easily be changed if it does not. Although not necessarily desirable, it is possible to change an ERROS application, even including changes to database structures, whilst it is in use and without affecting the response times of users. Once users are **satisfied with a prototype, it can be put into production as it is a working system**.



Network Nodes

As explained earlier, all data records are stored within their context of entity type number, entity number and attribute number, to which is added the unique attribute identifier to make a key field that uniquely identifies them. Since the key field does not expand at lower levels in an hierarchy, it is always possible to create a relationship at a lower level with any existing data record. This will in turn have a unique identifier and another relationship can be created with that, and this can carry on almost without limit. So, for instance, a company will have an employee and this connection is stored as a relationship. A lower level relationship could be to the department in which he works – he can only be a member of a department because he is an employee of the company. At a level below that might be his job function – he could only be a book keeper if he is in the accounts department.

Each relationship is a connection point or node between other nodes in the network. The menu that provides access to a node determines what action can be taken by the operator, how any records accessed are to be processed, and whether there is a lower level menu that can be accessed. However, users will not generally be aware of the concept of nodes.

Multi-dimensional Navigation

A node may store a relationship and, subject to security, operators will be able to navigate to the related record. For instance, a sales order will contain a list of products ordered. The operator can navigate to the record for a product and look at other orders for that product, or at manufacturing or supplier or pricing history. However, there may be lower level data about the relationship, such as the anticipated or actual delivery date for that product for that order if the whole order cannot be delivered in one shipment or perhaps the reason for the delay.

At any node in an ERROS application, there can be related data and also, where appropriate, lower level data about the relationship, such as the provenance of the data. There will always be an audit trail entry stating who put up the data above and when and in which application.

Cataloguing

ERROS applications can catalogue multiple collections of any type of object to create a "Union Catalogue". Multiple businesses or institutions are able to contribute to the general history and also able to catalogue their collections of any type of object in one, potentially massive, integrated database. The system was designed from the beginning for **multiple users from multiple institutions to share "public" data**, accessible by all, with "private" data only accessible by operators employed by the institution that owned the data, with the public and private data being **seamlessly integrated** from the viewpoint of the owner of the private data.

This means that when cataloguing objects of which there can be multiple examples, such as prints, books, coins, motor cars, stamps, etc., a standard description, bibliography, etc. for an object can be created by the first person to catalogue that object and subsequent cataloguers only need to add the accession (stock) number of their copy, its location, etc. But since this data will be private, users from other institutions will not know that it exists. Co-operative working dramatically reduces the cost of data entry. The same mechanism can be used to allow users to store their own private data, such as product prices or research notes, in a multi-institution database, making ERROS very suitable for any co-operative project, in business as well as in the arts and humanities.

Security

A major issue in all applications is security. Security can be controlled in ERROS **down to the individual attribute level**. Where an attribute contained a single field, security would be at the field level. This is achieved through relationships. There is an entity type "Operator



Identification”, also with the attribute “name and number”. Each operator's ID is retrieved when they log on, together with their job function and department if these are recorded. Each operator's record is related to the applications to which they are authorised. As the operators do not have direct access to the entity type “Application”, the related applications, which they operate by navigating to them, are the only ones to which they have access, ensuring security. This method of authorisation can be used throughout an application, or authority can be granted to employees with specific job functions only or to members of a department, etc.

ERROS adds considerably to the level of security provided by the IBM i operating system. In ERROS, when a user selects an application, his security levels, which are stored as part of the relationship between the operator's record and the application, are set for that application. There are **over twenty security values** set for permissions for access, add, relate, change, delete, remove, print and many more. Default values can be used. These are numeric values from 5 (highest security) to 9. Values from 0 to 4 are reserved for future use. Similar values are set in menu records. If the user has a value for function that is less than or equal to that set in the menu record, then he or she can execute the function.

Individual data and menu records can be given higher level security for access, change, delete and remove. This numeric value is deducted from the equivalent value in the preceding menu record. If the result is less than the operator's value for the application, then he or she will not be able to access the record or will not be able to undertake the function. This allows security to be set for an individual attribute iteration, whether it is a data record or a menu record. As a result, some users may not be able to see the record and won't realise that it exists. Senior management can be given a higher level of access authority yet perhaps less authority for entering or changing data in some financial applications, such as accounting.

The Connectionist Database

The database can be called a Connectionist Database, as it is **defined by connections that define other connections**. As ERROS combines the database and application environments and includes a standard operating interface, it provides a Connectionist Development Framework with which to build applications. ERROS can be used to create advanced applications with very complex data structures for any type of domain in any business or institution.

This structure is very different from anything else that existed then or now so it may seem complex, but, in practice, as it treats all data and definitions in exactly the same way, it is really quite simple. The detailed design only took about three weeks.

The **main programs**, which are used for data and application definition and for operating those applications, today are together **under 70,000 lines of code** and compile to **under 16Mb**. This includes code for the Database Handler, the interface and all the enhancements to date, listed below. The first version was somewhat smaller.

The challenge in my design was that the database would be defined in itself by itself so I couldn't create it until I had created it! However, by doing all the coding myself so that I didn't have to write a detailed specification for someone else to interpret, I managed to overcome this with iterative development. My original aim had been to build a fine art and history application, so I was now able to use ERROS to create this. We called it STIPPLE.



The ERROS Standard Operating Interface (SOI)

It seemed sensible at the beginning of development to separate the code for an interface from that for the database handler. The **interface could be used for all applications**. This meant yet another further enormous improvement in performance, as the same main programs are used by all users and remain in main memory, whatever the application. These programs are used in conjunction with the exit programs referred to earlier.

There is little difference in ERROS between creating, changing and operating applications, since the same methods and computer programs are used to update the same knowledge base. The ERROS Standard Operating Interface (SOI) requires minimal training and ensures that **operators always use the same consistent, easy-to-understand procedures**, whatever their task. SOI is designed to **keep screens uncluttered** and only display information relevant to the task in hand. Different operators can carry out the same task at the same time in the ways that best suit them. They can bypass menus and access any information with just one command or browse through the menus to find the data and procedures that are available to them.

ERROS applications operated using SOI can be operated **using "point and click" techniques or using standard function keys**. Point and click works on both 5250 terminals and when using a standard browser, as do the function keys. When accessed from a browser, ERROS generates the necessary HTML and Javascript code. **There are no static web pages in ERROS.**

The ERROS SOI works with all ERROS applications. Users could create their own standard interface if they prefer and use this instead of SOI.

Although standard HTML options set within ERROS are used by default, special user options, such as text and background colours and default font size, can be stored in the ERROS database and are retrieved when the user signs on.

The standard interface already generated **PCL 5 for printing**. ERROS now generates **HTML and JavaScript on the fly** so that all ERROS applications can be used over the Internet with most standard browsers, on Pcs, Macs, tablets and smart phones. There are no static HTML pages in ERROS, although links to external pages can be stored in ERROS.

Changing the structure of data will automatically change the look of the web pages.

Menus also define whether there are to be line breaks between each attribute iteration and whether the names of the attribute should actually be displayed on complex web pages or printed reports. Data from different attributes can be displayed together without a line break if required. Users will not necessarily be aware that what they regard as a single "logical" record is in fact the combined display of multiple physical records. This flexibility does not affect the very rapid response times of ERROS applications.

The ERROS Output Description Language (ODL)

The ERROS Output Description Language (ODL) is a powerful language that allows ERROS application developers to **define printed reports**, such as letters, documents, reports and manuals with a high degree of flexibility and allows polished presentation of the browser screen or printed page. **ERROS inserts the necessary HTML and JavaScript or PCL5 code into the datastream.**



Publishing on the Web

The ERROS Standard Operating Interface allows users to put **banners and logos** on their websites. Since the total ERROS production database and all ERROS applications that access it can be operated on the internet, the production database can also be the Internet database. There is no need to export data for publication on the web. External users can be given access to a limited range of applications, perhaps with read only security. However, this may not be desirable.

Another approach could be to use the high availability option of ERROS, with a second system maintaining a totally up to date copy of the database. External users who only require access could use this second system, reducing the workload on the main system.

ERROS screens

A range of ERROS browser screens are shown below. The HTML and JavaScript were all generated by ERROS on the fly, and the screens show the extraordinary flexibility of SOI. The applications can also be operated on a traditional 5250 terminal if required without any change to them being needed.

If the structure of the data is changed, the layout of the screens will change automatically.

ERROS Application Selection

After you sign on, ERROS will display the list of applications to which you are authorised.

Enter JUMP BACK Previous Refresh Audit View Help Print Operate Options Logout
ERROS Workgroup Organiser (An ERROS Application) Rob Dixon Rob Dixon Collection (ERROS08)

Rob Dixon
ERROS application(s) authorised to employee direct

Enter Repeat Input Add

- 1 Corporate Organiser II
- 2 ERROS Application Creator II
- 3 ERROS Authority Organiser
- 4 ERROS Business Modeller
- 5 STIPPLE IV

Earlier Records More

Product Inventory Screen

Enter JUMPBACK Previous Refresh Audit View Help Print Operate Options Logout
 Stock Level Adjustment (An ERROS Application) Rob Dixon Erros plc. (ERROS08)
 Initial Menu FastPath

product
 name and/or number

 # (Product) 0
 Enter Repeat Input Add

| | Cur ID | Units | Total Units | Sales | Purchase | Int.Ord | Int.Ord | Ret.to | Ret.by | Loans | Loans | Units | Future |
|----|--------|-----------------------------|-------------|------------|----------|---------|----------|--------|--------|-------|-------|-------|--------|
| | Sts | # | Total | Cost | Avail | Outs | Expected | Due | Out | Due | In | Supp. | Custom |
| | | | | | | | | | | | | | |
| 1 | 2 | Drawer metal filing cabinet | # 13 | 5 GBP | 123.67 | 5 | | | | | | | 5 |
| 2 | 3 | Drawer metal filing cabinet | # 8 | F Y 11 GBP | 230.45 | 6 | 2 | | 1 | | 1 | | 7 |
| 3 | 4 | Drawer metal filing cabinet | # 9 | 16 GBP | 483.61 | 15 | 1 | | | | | | 15 |
| 4 | A4 | Paper, ream, 100 gsm | # 12 | 60 GBP | 75.23 | 40 | 20 | | | | | | 40 |
| 5 | | Board Room chair | # 4 | 12 GBP | 1390.12 | 10 | 2 | | | | | | 10 |
| 6 | | Board Room table | # 7 | 3 GBP | 1897.00 | | 3 | 4 | | | | | 4 |
| 7 | | Economy typist's chair | # 6 | D 17 GBP | 345.76 | 7 | 10 | | | | | | 7 |
| 8 | | Executive chair | # 3 | 6 GBP | 701.54 | 6 | | | | | | | 6 |
| 9 | | Executive desk | # 5 | 2 GBP | 1501.93 | 2 | | | | | | | 2 |
| 10 | | Inkjet printer A4 | # 10 | 3 GBP | 35.23 | 2 | 1 | 3 | | | | | 5 |

Earlier Records More

The column widths will **expand automatically** if the values are larger. Selecting any one of the records will lead to other information about it, such as orders received, etc.

When using **5250 terminals or emulators**, a screen like the one above will not fit in a standard 80 character wide display, so ERROS automatically enables F20 (window right) and F19 (window left) in all ERROS applications in 5250 mode when required, allowing a line length of up to 356 characters.

All records in the table above are **indexed by name and by product number** and you can access any record by typing in the first word (or more) of its name (e.g. "board" for the products on lines 5 and 6) or a generic search term such as "E*".

- Typing "*" and pressing ENTER will display the records in product name sequence from the beginning of the table.
- Typing "*" and pressing or clicking on Page Up will cause ERROS to display the records in product name sequence, going backwards from the end of the table.
- Typing "#*" and pressing or clicking on ENTER will cause ERROS to display the records in product number sequence from the beginning of the table.
- To access a record by product number (shown to the right of the product name), you can type, for instance, "#12" in the main box or type "12" in the number box.
- You can use relative operators "<", "<=", "=", ">=" and ">" in front of any of the search terms, whether name or number, in the main input box.

Because **every record is indexed**, response times are **extremely rapid**. Even if there were, say, one million different products in stock, ERROS can find any record in a fraction of a second. Only enough records to fill the screen are sent to the browser or 5250. You can page down or up for others or make another request.

Some businesses sell low volumes of high value products and others sell high volumes of low value products. As part of the ERROS framework, a record can be stored for each ERROS customer company defining the maximum number of digits and decimal points for values, for prices and quantities. It also stores the accounting currency, VAT rates and the date from which they applied, on a country by country basis, VAT scheme, etc. ERROS automatically



expands the column display widths in both browser and 5250 mode. The above screen allowed for 5 digit quantities. With these and other similar features, software packages can easily be tailored to suit individual businesses or divisions within a business without any program change being required.

As a business expands, it may need to handle an ever increasing number and range of products. Finding a product if you don't know its product number and are not sure of its name can become a problem.

It is simple in ERROS to **categorise products into hierarchies**, such as Office furniture/chairs. This can be done at any time, without programming and whilst the application is live, without changing or disturbing the existing product records.

A continual problem with most applications is providing access to associated or connected data, solved either by new program coding for each link or perhaps by using SQL. A solution may have to be created on a table by table basis. **With ERROS, no action is required as, because ERROS is based on its connectionist database, ERROS knows where to find the related data.** Security can be set to prevent this navigation if required.

Orders Received Screen

Double clicking on, say, *A4 paper* (item 4 on previous screen) will immediately cause ERROS to display the following screen -

product name and/or number
A4 Paper, ream, 100 gsm #000000012

Quantity 0 Price 0

| | Quan. | Price | Incl. VAT | Quan. | Total | Due |
|----------------|-------|-------|-----------|-------|-------|-----|
| 10/01/13 12.24 | 20 | 4.00 | 80.00 | 20 | | |
| 24/01/13 17.37 | 10 | 4.00 | 40.00 | 10 | | |

This shows, in date sequence, all orders for the product *A4 Paper, ream, 100gsm*. No SQL or any other type of query was necessary to achieve this.

ERROS shows, in all screens, above the (large white) input box, the context of the data retrieved and, below the input box, the data retrieved in that context. The Stock Level Adjustment application defines how ERROS is to display the data.

The Add button is not active and the Edit and Delete buttons are not displayed so the operator cannot change the data.

Throughout the screens in this Stock Adjustment Application, attribute iterations displayed in yellow represent connections that can be navigated.



The layout of the detail lines in this Sales Order is different from those for products in the previous display. With traditional application development methods, a separate solution has to be created for each table for:

- accessing the records,
- displaying them,
- ensuring that the solution can handle the situation when there are too many records to fit on a screen,
- providing a means of finding individual records in the table,
- allowing navigation to connected data, etc.

Whilst this may not be particularly technically demanding in a traditional environment, it is time consuming and error prone and means that many new programs have to be created as well as SQL. These all need programming, testing, documenting and controlling.

On the IBM i, many solutions would include the use of sub-files defined in DDS to display the detail lines, with a separate sub-file being defined for each record type. This again is time consuming. In ERROS, there is just one standard ERROS mechanism for this. This doesn't use IBM i sub-files even though it may look the same. This standard mechanism is used for all detail line displays in all applications. When adding new types of data in a traditional environment, new table objects are created **as these fit automatically within the existing ERROS framework.**

You can double click on one of the orders to look at that order as shown in the next screen. Alternatively, you can single click on an order and press F7 or click on the Audit button and ERROS will inform you who added/updated the order line and when.

Order Detail Screen

The screenshot shows the ERROS Order Detail Screen. At the top, there are navigation buttons: Enter, JUMPBACK, Previous, Refresh, Audit, View, Help, Print, Operate, Options, Logout. Below these are fields for Stock Level Adjustment (An ERROS Application), Rob Dixon, and Erros plc. (ERROS08). There are also buttons for Initial Menu and Fast Path. The main area displays a sales order for 10/01/13 12.24 Esmond Estates Ltd with order number #000000001. A search field is present with the value 0. Below the search field are buttons for Enter, Repeat Input, and Add. The main table lists items with columns for Item #, Description, Quantity, Price, and Inclusive Quantity. The items are:

| | Quan. | Price | Incl. Quan. |
|---|-------|-----------|-------------|
| | Ord. | Incl.VAT | Total Due |
| 1 3 Drawer metal filing cabinet | # 8 | 1 48.00 | 48.00 1 |
| 2 4 Drawer metal filing cabinet | # 9 | 1 56.00 | 56.00 1 |
| 3 A4 Paper, ream, 100 gsm | # 12 | 20 4.00 | 80.00 20 |
| 4 Board Room chair | # 4 | 3 80.00 | 240.00 3 |
| 5 Board Room table | # 7 | 1 1200.00 | 1200.00 1 |
| 6 Economy typist's chair | # 6 | 10 35.00 | 350.00 10 |
| 7 Executive chair | # 3 | 1 120.00 | 120.00 1 |
| 8 Inkjet printer A4 | # 10 | 1 80.00 | 80.00 1 |
| 9 Laser printer A4 | # 11 | 1 150.00 | 150.00 1 |
| 0 Stacking chair | # 1 | 10 40.00 | 400.00 10 |
| 11 Items EX VAT 2270.00 VAT 454.00 40 2724.00 40 OVERALL TOTALS | | | |

At the bottom, there are buttons for Earlier Records and More.

ERROS automatically navigates to the order of which the selected record was part.

The Add button is not active and the Edit and Delete buttons are not displayed and the operator cannot change the order which was created in an order entry application.



The first 10 order lines are displayed in the sequence of their product names. Typing "#*" and pressing Enter will result in the order being displayed in product number sequence. Any order line can be retrieved by product name or number in a fraction of a second, however many lines there are in an order.

You can double click again, on, say, Boardroom Table and ERROS will display the following screen -

Boardroom Table orders

Enter JUMPBACK Previous Refresh Audit View Help Print Operate Options Logout
 Stock Level Adjustment (An ERROS Application) Rob Dixon Erros plc. (ERROS08)
 Initial Menu Fast Path
 product name and/or number Board Room table #000000007
 #000000007
 # Quantity 0 Date 0 0 0 Time 0 0
 Enter Repeat Input Add

| | Quan. | Price | Incl. Quan. |
|--|-------|----------|-------------|
| | Ord. | Incl.VAT | Total Due |
| 1 10/01/13 12.24 Esmond Estates Ltd # 1 | 1 | 1200.00 | 1200.00 1 |
| 2 24/01/13 17.37 Thomas Linley Baths Ltd # 2 | 1 | 1000.00 | 1000.00 1 |
| 3 14/02/13 13.49 Boarstall Aggregates # 3 | 1 | 850.00 | 850.00 1 |

Earlier Records More

Sales Order Entry

The next screen is for a sales order in a Sales Order Entry application. In this application, the operator has authority to change, delete or add to the lines in this order so the edit and delete buttons are now displayed and are active. Once order entry has been completed, then the order could be marked closed and this would no longer be possible. This order has been left open deliberately for demonstration purposes.

Enter JUMPBACK Previous Refresh Audit View Help Print Operate Options Logout
 Sales Order Entry (An ERROS Application) Rob Dixon Erros plc. (ERROS08)
 Initial Menu Fast Path
 Esmond Estates Ltd #000000032
 date and/or number 10/01/13 12.24 Sales order #000000001
 #000000001
 # Quantity 0
 Enter Repeat Input Add

| | Quan. | Price | Incl. Quan. |
|---|-------|----------|-------------|
| | Ord. | Incl.VAT | Total Due |
| 3 Drawer metal filing cabinet # 8 | 1 | 48.00 | 48.00 1 |
| 4 Drawer metal filing cabinet # 9 | 1 | 56.00 | 56.00 1 |
| A4 Paper, ream, 100 gsm # 12 | 20 | 4.00 | 80.00 20 |
| Board Room chair # 4 | 3 | 80.00 | 240.00 3 |
| Board Room table # 7 | 1 | 1200.00 | 1200.00 1 |
| Economy typist's chair # 6 | 10 | 35.00 | 350.00 10 |
| Executive chair # 3 | 1 | 120.00 | 120.00 1 |
| Inkjet printer A4 # 10 | 1 | 80.00 | 80.00 1 |
| Laser printer A4 # 11 | 1 | 150.00 | 150.00 1 |
| Stacking chair # 1 | 10 | 40.00 | 400.00 10 |
| 11 Items EX VAT 2270.00 VAT 454.00 40 2724.00 40 OVERALL TOTALS | | | |

Earlier Records More



The data definitions can ensure data integrity so that, for instance, an order header cannot be deleted leaving order details still there.

In the above screen, there is a total line that displays the values in the order header record. As order lines are added or changed, the order header in the database and its display on the screen will also change. In addition, the stock record for the product is updated as are other relevant records. In traditional systems, this might be done with trigger programs. There is no need for these in ERROS as the Sales Order Entry application definition instructs ERROS which records are to be updated. The operator will be unaware of these changes. Commitment Control is used to ensure data integrity. **If journalling is accidentally or deliberately switched off, ERROS will stop working.**

When you are using a browser, you can operate your ERROS application using point and click methods or you can operate it as though you were using a 5250 terminal, as the ERROS browser interface emulates 5250 style function keys that you can press and it also provides equivalent buttons that you can click.

By using the Options button (in the top line), you can instruct ERROS to display short cut keys on the buttons as in the following screen. The short cut keys are active whether or not they are displayed.

The screenshot shows the ERROS Sales Order Entry application interface. At the top, there is a menu bar with buttons for Enter, JUMP BACK (F3), Previous (F12), Refresh (F5), Audit (F7), View (F8), Help (F1), Print (F13), Operate (F15), Options (F16), and Logout. Below the menu bar, the application title is "Sales Order Entry - An ERROS Application" and the user is identified as "Rob Dixon" for "Erros plc. (ERROS08)".

The main area displays order details for "Esmond Estates Ltd" with order number "#000000032" and date "10/01/13 12.24". The sales order number is "#000000001". A search bar is visible with the value "0".

Below the search bar, there is a table of items with columns for quantity, price, and VAT. Each row includes a description, a quantity, a price, and VAT information. The table is as follows:

| | Quan. | Price | Incl. Quan. |
|---------------------------------|-------|-----------|-------------|
| | Ord. | Incl. VAT | Total Due |
| 1 3 Drawer metal filing cabinet | # 8 | 1 48.00 | 48.00 1 |
| 2 4 Drawer metal filing cabinet | # 9 | 1 56.00 | 56.00 1 |
| 3 A4 Paper, ream, 100 gsm | # 12 | 20 4.00 | 80.00 20 |
| 4 Board Room chair | # 4 | 3 80.00 | 240.00 3 |
| 5 Board Room table | # 7 | 1 1200.00 | 1200.00 1 |
| 6 Economy typist's chair | # 6 | 10 35.00 | 350.00 10 |
| 7 Executive chair | # 3 | 1 120.00 | 120.00 1 |
| 8 Inkjet printer A4 | # 10 | 1 80.00 | 80.00 1 |
| 9 Laser printer A4 | # 11 | 1 150.00 | 150.00 1 |
| 0 Stacking chair | # 1 | 10 40.00 | 400.00 10 |

At the bottom of the table, there is a summary line: "11 Items EX VAT 2270.00 VAT 454.00 40 2724.00 40 OVERALL TOTALS".

At the bottom of the screen, there are buttons for "Earlier Records (PgUp)" and "More (PgDn)".

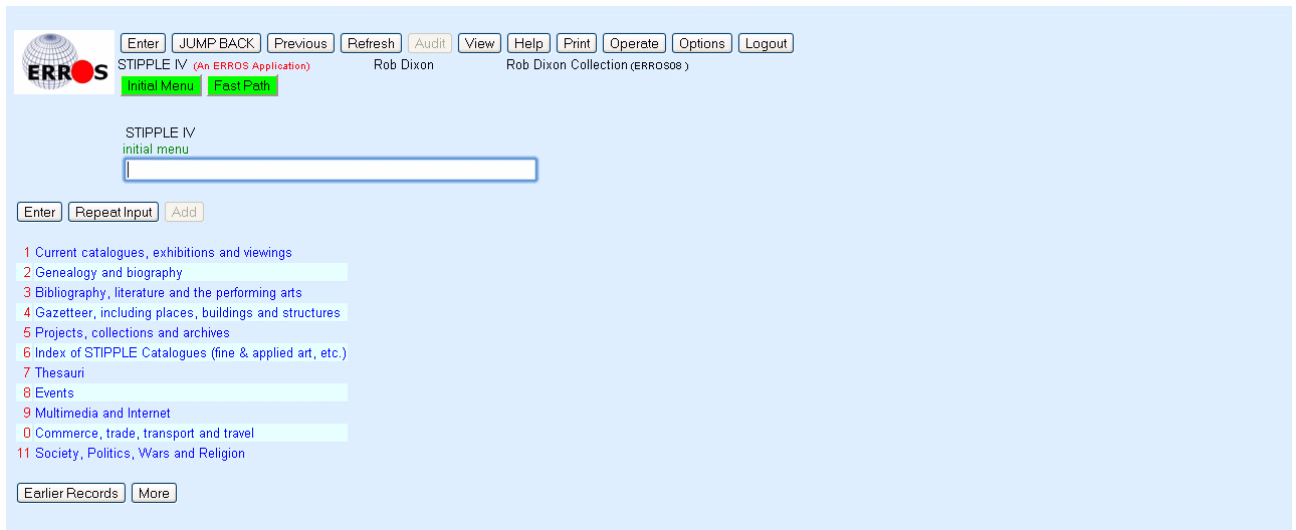


Smart phone screen



STIPPLE screens (The ERROS application for recording history)

Initial menu



Menus may be nested with many lower levels if required. **(Why did you take this out?)**

At any point in the application, the operator can return to this menu by clicking on the initial menu button towards the top left. There is also a fast path menu that allows operators to bypass the menu structure and access records with a single command.

Boarstall Tower

STIPPLE has a facility for recording places and buildings. The following display is for Boarstall Tower, in Buckinghamshire, England, the home of ERROS. Although the data is very different from the Product Inventory screen shown above, exactly the same ERROS mechanism that generates HTML and JavaScript on the fly is used in this and in all other ERROS screens.






[Enter](#) [JUMP BACK](#) [Previous](#) [Refresh](#) [Audit](#) [View](#) [Help](#) [Print](#) [Operate](#) [Options](#) [Logout](#)

STIPPLE IV (An ERROS Application) Rob Dixon Rob Dixon Collection (ERROSos)

[Initial Menu](#) [Fast Path](#)

Boarstall Tower #000000188
[name and/or number](#)
 Boarstall Tower #000000188
[initial display](#)

[Enter](#) [Repeat Input](#) [Add](#)

Boarstall Tower, Boarstall, Buckinghamshire, England, listed building, Grade I, banqueting pavilion (possibly), fortified building, gatehouse built 1312, house converted 1925, hunting lodge upgraded 1615
 open to public
 19/3/14 24/09/14 Wednesdays 14.00-17.00 [last admissions 30 minutes before closing](#)
 19/4/14 25/08/14 Bank Holiday Mondays 11.00-17.00 Saturdays on Bank Holiday weekends only 11.00-17.00 [last admissions 30 minutes before closing](#)
 The "superb C14 gatehouse" (listed Grade I), and gardens with large moat, of Boarstall House (demolished 1778).
 Sir John de Haudio (of Boarstall) built "Buckinghamshire's only complete medieval fortified building" in 1312, both as defences for Boarstall House and as a grand statement of his status. Although updated in 1615 for use as a banqueting pavilion or hunting lodge and to reflect the latest taste, including "handsome" oriel windows, and the upgrading of its "fine" top floor banqueting chamber, it retains its medieval floor plan, its medieval belfry, crossloops, crenellations and other features, so keeping its fortified look, still fashionable in Jacobean times. Today, the exterior and many rooms remain virtually unchanged from then, despite considerable fighting in the British Civil War when Boarstall changed hands three times, resisting a siege by 1200 men from the New Model Army under Sir Thomas Fairfax in 1645, and finally surrendering after a ten week siege in June 1646.
(Quotations taken from "Buckinghamshire" by Nicolaus Pevsner and Elizabeth Williamson, 2nd. Ed., Penguin 1994. ISBN 0 14 0710 82 0)
[main features of building](#) [banqueting chamber](#), [hexagonal turrets](#), [moat](#) and [oriel windows](#)

There are 29 navigable links, including the images, on the above page. There can be any number of links or images. The Google Map works just as if you had retrieved it in Google, even though it is part of the STIPPLE display. Pressing Enter you find further information about Boarstall Tower.

A Single ERROS Print

[Enter](#) [JUMP BACK](#) [Previous](#) [Refresh](#) [Audit](#) [View](#) [Help](#) [Print](#) [Operate](#) [Options](#) [Logout](#)

STIPPLE IV (An ERROS Application) Rob Dixon Rob Dixon Collection (ERROSos)

[Initial Menu](#) [Fast Path](#)

The Right Hon.ble Lady Betty Delmé. #000035356
 Rob Dixon Print Collection #000000288
[print details](#)
[summary information with image](#)

[Enter](#) [Repeat Input](#) [Add](#)



The Right Hon.ble Lady Betty Delmé. #35356
[mezzotint](#)
[Valentine Green](#) after Sir Joshua Reynolds P.R.A.
 1595 x 385, PI 626 x 385 mm (RSD P # 288)
 Published 1/7/1779 by Valentine Green; 4/1790 by William Richardson; c 1890 by P. & D. Colnaghi and Co.
 Portrait of Lady Elizabeth Garnier (olim Delmé, née Howard), 1747 - 1813, daughter of Henry, 4th Earl of Carlisle, WL, John Delmé, 1772 - 1800 and Miss Isabella Delmé, c 1774 - 1794 WL
 Our impressions Rob Dixon Print Collection RSD P # 288; RSD P # 1375; RSD P # 1376
 Published as part of a set or series of *Beauties of the Present Age* Pl. 6 (although engraved first)

The above screen shows the preliminary information about a mezzotint engraving of the Lady Betty Delmé. There are three copies of this print (all different) in the Rob Dixon Collection. The name of the collection is only displayed once for each print in a collection, making the screen easier to read. STIPPLE can handle multiple collections. The information about Betty Delmé and her children is not stored in the record for the print but is automatically retrieved from their individual biographical records. Thus this data only needs to be stored once, however many prints there are of each sitter, and the data displayed, apparently as part of the print record, is totally consistent from print to print.



The last line of text states that the print was "Published as part of a set or series of *Beauties of the Present Age*" as Pl. 6. Double clicking on this link will lead to the display of the set as shown below.

A Set of Prints in STIPPLE

Unlike many other systems. ERROS applications handle sets and groups of records with ease. The screen below shows the first 6 prints from a set of 11, the print of Lady Betty Delmé being the last print displayed on this screen. To display further prints in the set, the operator presses Enter.

STIPPLE IV (An ERROS Application) Rob Dixon Rob Dixon Collection (ERROS08)

Initial Menu Fast Path

Enter Repeat Input Add

Beauties of the Present Age #37
A set of eleven mezzotint whole length female portraits after Sir Joshua Reynolds, engraved and published by Valentine Green.
contains

| | | |
|---|---|---|
| <p>[main record] Pl. 1 (1st pair)</p> <p>The Right Hon:ble Lady Louisa Manners, Sister to the Earl of Dysart. #37016 mezzotint Valentine Green after Sir Joshua Reynolds P.R.A. 1595 x 382, Pl 631 x 384 mm (RSD P # 179) published 24/12/1779 by Valentine Green, 1/1/1790 by John Brydon portrait of Lady Louisa Manners Countess of Dysart (née Tollemache) Our impressions - RSD P # 179</p> | <p>[main record] Pl. 2 (1st pair)</p> <p>The Right Hon:ble Lady Jane Halliday, Sister to the Earl of Dyfart. #35458 mezzotint Valentine Green after Sir Joshua Reynolds P.R.A. 1597 x 386, Pl 630 x 386 mm (RSD P # 1278) published 24/12/1779 by Valentine Green portrait of Lady Jane Ferry (olim Halliday, née Tollemache) Our impressions - RSD P # 1278</p> | <p>[main record] Pl. 3 (1st extension)</p> <p>Jane, Countess of Harrington. #5547 mezzotint Valentine Green after Sir Joshua Reynolds P.R.A. 1596 x 387, Pl 630 x 388 mm (RSD P # 1279) published 1/5/1780 by Valentine Green, ç 1890 by P. & D. Colnaghi and Co. portrait of Jane Stanhope, Countess of Harrington (née Fleming) as Aurora Our impressions - RSD P # 1269</p> |
| <p>[main record] Pl. 4 (2nd pair)</p> <p>Mary, Isabella, Duchess of Rutland #5994 mezzotint Valentine Green after Sir Joshua Reynolds P.R.A. 1598 x 385, Pl 631 x 386 mm (RSD P # 174) published 1/7/1780 by Valentine Green, 1793 by Charlotte Brydon portrait of Mary Isabella Manners, Duchess of Rutland (née Somerset) The head on this plate was altered in 1793 to represent the Duchess of York. Our impressions - RSD P # 174; RSD P # 1333</p> | <p>[main record] Pl. 5 (2nd pair)</p> <p>Georgiana, Duchess of Devonshire #432 mezzotint Valentine Green after Sir Joshua Reynolds (P.R.A.) 1599 x 385, Pl >633 x 385 mm published 1/7/1780 by Valentine Green portrait of Georgiana Cavendish, Duchess of Devonshire (née Spencer) Our impressions - RSD P # 323; RSD P # 1211; RSD P # 1334; RSD P # 1468</p> | <p>[main record] Pl. 6 (although engraved first)</p> <p>The Right Hon:ble Lady Betty Delmé. #35356 mezzotint Valentine Green after Sir Joshua Reynolds P.R.A. 1595 x 385, Pl 626 x 385 mm (RSD P # 288) published 1/7/1779 by Valentine Green, 4/1790 by William Richardson, ç 1890 by P. & D. Colnaghi and Co. portrait of Lady Elizabeth Garnier (olim Delmé, née Howard); John Delmé, Miss Isabella Delmé Our impressions - RSD P # 288; RSD P # 1375; RSD P # 1376</p> |

This screen has 56 navigable links (in blue). Further information about any individual print can be accessed by clicking on "main record" as shown on the previous page. Although this display is taken from a catalogue of sets of prints, the only data taken from the set catalogue are the name of the set, the brief description, the attribute "contains" and the sequence of the prints in the set. The details of each print are retrieved from the print catalogue so these are only stored once. If details of a particular print change, then this display will change, even though the data in the set catalogue has not been changed.

Multiple Record Identifiers

Names of people, products and much else change. It is vital that operators can access a record by its previous name, otherwise, if they don't realise that a name has been changed, they may assume that the record does not exist and create another which is in fact a duplicate. ERROS allows synonyms at any point so that any records identified by their current name can still be accessed under their earlier names.



Finding Data in Google and ERROS Applications

The World Wide Web now stores vast quantities of data, much of it replicated many, many times. This makes retrieval of data using a search engine somewhat problematic. Typing "building" in Google produces 550 million pages at the time of writing. A searcher cannot begin to look at any more than a very limited part of these.

A major flaw in the structure of the Internet is the lack of bi-directional relationships, a standard feature of all ERROS applications.

In STIPPLE, you will find that building is an entity type, and a list of buildings is accessible in name sequence, each with a unique number in case there is more than one building with the same name (as is sometimes the case). If you want to restrict your search to bridges, you can access "bridge" in the entity type "type of building" and then get a list of all bridges. If you want to restrict your search to buildings in a particular town, such as Aylesbury in Buckinghamshire, England, you start with Aylesbury and can look for all buildings or restrict your search to a particular street in Aylesbury.

Retrieving people by their names is another problem when using Google. Lady Elizabeth Howard was a daughter of the 4th Earl of Carlisle (Howard was his family name). She married Peter Delmé, M.P, so became Lady Elizabeth Delmé. Her husband was not titled, but, as the daughter of an earl, she could continue to use her own title. She was generally known as Betty and a very famous print of her, published in 1779, was given the title *The Right Hon.ble Lady Betty Delmé*, shown earlier. After her first husband died, she married a naval officer, Captain Charles Garnier, so she became Lady Elizabeth Garnier, retaining her own title.

Google retrieved 10,4 million pages for Lady Elizabeth Howard, 119,000 pages for "Lady Elizabeth Howard", 343,000 pages for Lady Betty Delmé, 41,000 for "Lady Betty Delmé", 4,420 pages for Lady Elizabeth Delmé (yet 2 minutes later 4,340), 21,700 pages for "Lady Elizabeth Delmé" (why there are more pages when quotes are used is not obvious), 1.77m for Lady Elizabeth Garnier, just 5 for "Lady Elizabeth Garnier", and none for "Lady Betty Garnier". Yet there was, as far as I am aware, only one Lady Elizabeth Garnier.

In STIPPLE, there is only one record, under her final name Lady Elizabeth Garnier, and, using synonyms, this can also be accessed under Betty Delmé, Elizabeth Delmé, Betty Garnier and her maiden name, Elizabeth Howard. However, her record can also be accessed through relationships with records for her parents, her two husbands, her two children, and two prints and an oil painting of her. If the houses at which she lived were recorded in STIPPLE, then her record would also be accessible through date dependent relationships with those buildings. STIPPLE allows duplicate names, each uniquely identified. There can be many John Smiths. Traditional environments, and Google, find it very difficult to overcome these problems.

ERROS does not have the problems that arise with search engines and it might be said that the Internet is drowning because of its incredible success, because of the enormous volume of redundant data. If an ERROS based application, such as STIPPLE, were used to store the same data, without redundancy, these problems would not arise.

Archiving

Archiving can mean the cataloguing of large historic, probably paper based, archives and ERROS is ideal for this.

In database terminology however, archiving can mean being able to **save and then retrieve an earlier state of a database** – for instance when a person's address is changed and there is a need to find the old address after it has been deleted. It can also mean moving old, apparently redundant, data to a secondary storage device.



Delete states

ERROS allows two "delete" states – in the first a record is **tagged as deleted but not removed** from the database. Because ERROS allows all attributes to have multiple instances, when, for instance, someone's address is changed, the old record is not updated in place but is tagged as deleted and a new record for the new address is added. Tagging a record as deleted is the normal method of deletion in ERROS. The deleted record can be retrieved, and perhaps re-activated, by a user with adequate authority. Multiple instances mean that a person can change his or her address very many times without problem, with all old addresses being retained. Multiple instances also allow people and businesses to have multiple addresses.

The second method of deletion is the **physical removal of the record** from the database. Authority for this is rarely granted to any user. **Such a record can only be recovered by rolling back the database.**

Since ERROS is designed for "Big Data" and has very rapid performance that does not degrade noticeably as data volumes grow, having large numbers of "historic" records is not an issue unless there is a shortage of disk space. Today additional disks are not expensive. As there is **no performance overhead, leaving these records in place** may be a much simpler, cheaper and more controlled solution than archiving them outside the system, keeping track of where they are, and having to retrieve and perhaps restore them.

Users may want to know the state of the database at an earlier date. ERROS supports roll back but this may not be the best way of finding historical data. In an historical database, unlike business databases, data tends to be entered in a somewhat random sequence. Users don't want to be able to roll back the database to a particular date as that date might be a long time before the database was established. Instead they need to know what data was valid on a given date. ERROS allows date dependent relationships to be stored, so that the history of, say, a person's addresses can easily be seen. Date dependent relationships can be an equally powerful function in modern business databases. It is possible to enter a future address, or perhaps a price, etc. with the applicable start and end dates, before they become current. ERROS will select the one valid for the transaction date.

Data Integrity

Journalling and commitment control

When IBM added journalling to its integrated DB2 Universal Database, a long time ago, I used that to implement journalling in ERROS, with before and after images being recorded. This in turn allowed the implementation of commitment control, but with **commitment boundaries being defined in the ERROS database** rather than in programs.

The ERROS tables are always journalled – **if journalling is switched off, ERROS will not work**. Since new entity types are added to these existing tables, they will automatically be journalled and backed up.

A standard function of the DB/2 database that is integrated with the operating system is that, if there is a system failure, and a transaction defined by commitment control boundaries has not been fully posted to the database, then, after restart, users cannot sign on until the problem has been resolved, either by completing the transaction if all the data had been entered, or rolling back the incomplete transaction and informing the user.

Backup and Recovery

All changes to the business model, to the application definition, to security and to user data are always journalled with before and after images being recorded. These images are stored in a journal receiver. The ERROS System Operator Tasks application includes a non-stop program that backs up the journal receivers, any changed programs and all changed PC type files at



regular intervals to disk and then to immediately to tape. This is normally done hourly but the interval can be set by the system operator.

The ERROS System Operator Tasks application also includes facilities for both a daily full backup and a weekly full backup of all data apart from the operating system. There is also a system back option to back up the operating system.

The IBM hardware, database and operating system are very robust, and as the system has Raid disks, with hot swap, there should be very few occasions when a total system failure occurs and the system has to be restored. A full back up would normally be done at weekly intervals, but the daily backup option is there if required.

The system would be reinstalled by restoring the last full backup and then applying all the changes since then that have been saved in the journal receivers. This is a straightforward process.

Referential Integrity

ERROS menus can be used to ensure referential integrity so that, for instance, no record can be deleted if there are lower level or dependent records that have not been deleted or removed. Alternatively, ERROS can be set up to ensure that dependent links are either tagged for deletion or removed automatically.

Most, if not all, other databases, such as DB2, can be updated by using SQL or another utility. Any database should only be updateable by an authorised user of a proper application. In an ERROS environment, no user should have authority to SQL or any utility that can change data outside controlled procedures. If a user tried to update the ERROS files using a utility, he would have no idea what he was doing. All he could do was inflict random damage. Fortunately, this would be journalled, and could be rolled back to restore the earlier state. Journalling cannot be switched off whilst ERROS is running. If journalling is switched off, ERROS will not start.

ERROS High Availability Option

ERROS has high availability functionality added, so, if implemented, this is automatically a function of all existing and all future ERROS applications, allowing one or more duplicate copies of the database anywhere in the world to be synchronised with the production version. All database, application and ERROS security definition changes and all user data changes are copied to the target system. This is normally set up for asynchronous processing, so, if the second system is being backed up or communications to it fail, the prime system will continue working, and as soon as the target system is available again, the delayed transactions will be applied until it catches up.

Transaction Processing & Concurrency Control

Transactions in ERROS applications are defined in menu records in the database. These determine the records to be added, updated or deleted, and also the commitment control boundaries that define the beginning and end of a transaction. As all database changes are journalled (with both before and after images being recorded), if a transaction fails for any reason, the changes will be rolled back to the previous transaction boundary and the operator will be warned. The audit trail will record that the transaction failed.

If an operator displays a record for update on his screen, that record is not locked at that point so anyone else can still read it. Instead **concurrency is automatically controlled by saving the last transaction number** that affected that record and at update time, after the operator has committed the update, re-reading the record and locking it and checking that the



transaction number has not changed. If the transaction number has changed because another transaction has updated the record whilst the operator had it displayed on his screen, then his transaction will be rolled back and he will be informed. If the record has not been changed, then it is marked as a pending update or delete or, if it is a new record to be added, then it is added and marked as a pending add. The remainder of the transaction is completed, and, as the last step in the transaction, the pending marker is removed for an add or an update, or a delete, or, for a remove, the record is finally removed. The pending marker prevents the record from being changed by another operator until the transaction has been successfully completed.

Where a transaction adds, or updates or deletes a detail record but also updates a header record, as in order processing, ERROS also saves automatically the transaction number of the header record without locking it, and then locks and checks the transaction numbers of both records when the operator commits the update. If either has changed, the transaction will be rolled back, and the operator will be informed.

This method of concurrency control gives the optimum performance as record locking will only occur for a very short time. As concurrency control is a standard feature of the ERROS database handler, developers do not have to be concerned about it.

Responses for transaction processing are very rapid.

Performance of ERROS Applications

Many aspects of the ERROS design ensure outstanding performance. These include –

- No constant opening, security checking and closing of files
- Security of entity types controlled within ERROS – very much quicker
- No constant calling and freeing of a different program for each file
- No constant calling and freeing of different programs for each application
- A single database handler and a standard interface program are used at all times for all purposes, whether defining the data model, defining the application or operating the applications. Whatever the number of users, only one copy of each program needs to be in main memory and can remain there. Different programs are not constantly being paged in and out of main memory.
- The response times are not dependent on the database design or the skills of the developer and are totally predictable and consistent for almost all applications
- The database design ensures scalability – response times do not noticeably degrade as data volumes grow
- There are no database joins and navigating relationships is almost instant as no files have to be opened or closed
- No Null values
- It is possible to access the values for a single attribute without having to read the whole logical record for an entity
- No SQL is used
- Minimal record locking for concurrency control
- All ERROS applications can be run in main memory for maximum performance for very high usage
- It is possible to find the hierarchies to which a record belongs without searching any of the hierarchies
- the ERROS database is both read optimised & write optimised at the same time

In addition, the IBM operating system spreads the data across the available disks and disk arms and balances the load on them.



The AS/400 (now IBM i) laboratory experts in Rochester, Minnesota were very surprised when they saw – with an early version of ERROS - that the **response times of ERROS applications were about one tenth of those that they expected from similar, traditionally created, applications.** Before I went to demonstrate ERROS to IBM, they insisted that I patented the concepts of ERROS and this I was able to do.

ERROS Today

Of course, ERROS has evolved considerably since the early days. A trigger function was defined in the database, that doesn't require any special trigger programs. These changes allowed the implementation of transaction processing in ERROS. A transaction number had been allowed in the design so an audit trail was built. This records, for every change to data, application definitions or user data, who had made the change, when, and in which application. Since all data and application definitions are stored in the database, any changes to these can be rolled back as can changes to user data. Journalling and the audit trail together mean that every change to definitions is fully documented automatically to a consistent standard that programmers rarely achieve in traditional application development methods. For maximum performance, the ERROS database can be run as an in-memory database with all data in main memory at runtime.

STIPPLE, the application for recording history and for cataloguing my friend's and my print collections, has also evolved over the years, shown earlier. As originally anticipated, the structure of the STIPPLE database is now extremely complex. I experimented with analogue images using laser disks in the late 1980s, linking the laser disk images to the STIPPLE database via an IBM PC. The mechanism worked but the cost of creating the laser disks was just too high. Digital images have been available in all ERROS applications for many years.

I was lucky that I needed to catalogue prints rather than any other type of fine art object, such as paintings, which, superficially at least, may seem somewhat similar. Prints, however simple they may seem, are very complex objects to catalogue, and, if you have solved their problems, you have solved most computer database problems. Without prints and without the original System/38, ERROS, in the form of the modern Application Development Framework of today, would never have happened.

The structure of ERROS and its applications allows for enormous flexibility. Multiple developers can develop different parts of the same application concurrently, and all their entity type, attribute and application definitions will automatically be integrated, as well as the user data from the separately developed applications.

ERROS works extremely well in an expanding environment of largely unpredictable data – the real world. It would be ideal for creating ERP and other large systems (such as hospital or health authority systems, research projects, etc..). A group of software companies with specialist skills could develop separate parts of a major application, such as ERP, and these would all be integrated. ERROS can be used to create systems that require multiple operators from multiple institutions to contribute to a massive curated central database. All applications can easily be tailored to the needs of all users, generally without writing (or generating) a single line of program code. Costs and lead times are dramatically reduced, and the applications can be changed much more quickly and cheaply, in line with the ever evolving requirements of users as their markets and competition both change.

ERROS has continually evolved and today provides a connectionist development framework that dramatically simplifies the processes of application creation and maintenance. It can be called a truly modern method of application development, based on connectionist principles.



ERROS videos on YouTube

Videos that illustrate the creation of a very simple application are being put up on YouTube. A link to these can be found at

www.erros.co.uk/ERROS_YouTube.htm

