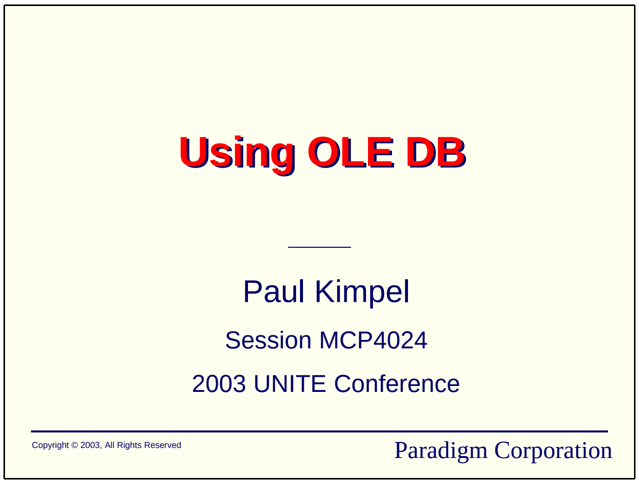


Using OLE DB

Paul Kimpel

Session MCP4024

2003 UNITE Conference

Copyright © 2003, All Rights Reserved

Paradigm Corporation

Using OLE DB

2003 UNITE Conference

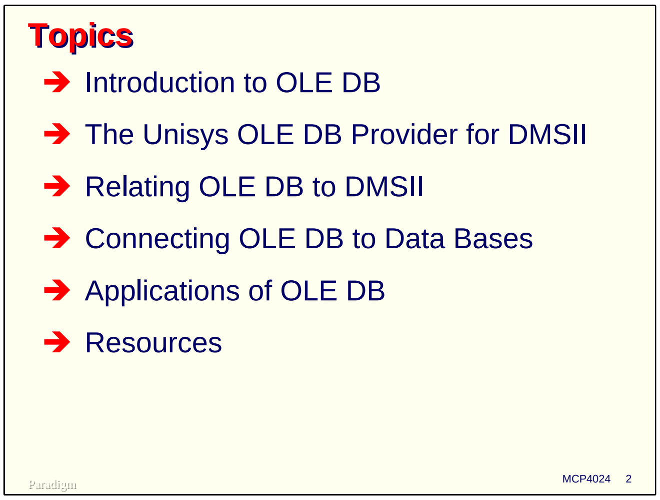
Reno, Nevada

Paul Kimpel

Paradigm Corporation
San Diego, California

http://www.digm.com

e-mail: paul.kimpel@digm.com



Topics

- Introduction to OLE DB
- The Unisys OLE DB Provider for DMSII
- Relating OLE DB to DMSII
- Connecting OLE DB to Data Bases
- Applications of OLE DB
- Resources

Paradigm MCP4024 2

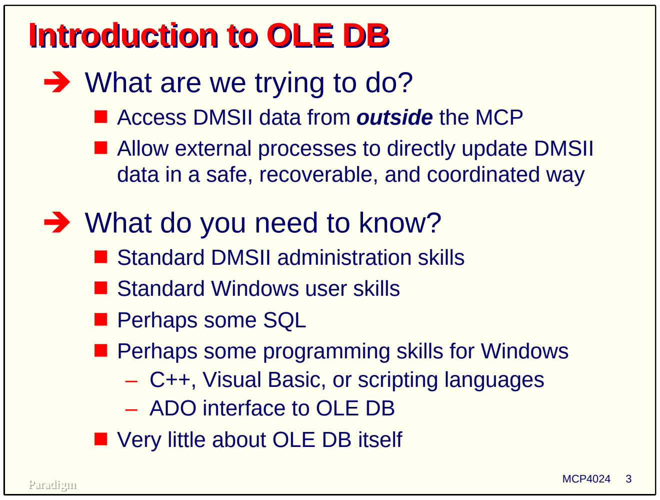
This presentation discusses the OLE DB Provider and its use under the Unisys ClearPath MCP.

I'll give a brief overview of OLE DB itself, then delve into the capabilities and features of the OLE DB Provider Unisys has developed for DMSII. Next, I'll discuss how OLE DB concepts relate to DMSII concepts and how DMSII features map to OLE DB ones.

To use OLE DB you have to connect to a data source, so the next section discusses connections to DMSII data bases.

OLE DB can be used by a number of tools and objects within the Microsoft Windows environment. I'll discuss the major ones in some detail and give an overview of several others.

Finally, I'll briefly discuss a number of resources where you can get more information and see sample applications.



Introduction to OLE DB

- What are we trying to do?
  - Access DMSII data from ***outside*** the MCP
  - Allow external processes to directly update DMSII data in a safe, recoverable, and coordinated way
- What do you need to know?
  - Standard DMSII administration skills
  - Standard Windows user skills
  - Perhaps some SQL
  - Perhaps some programming skills for Windows
    - C++, Visual Basic, or scripting languages
    - ADO interface to OLE DB
  - Very little about OLE DB itself

Paradigm MCP4024 3

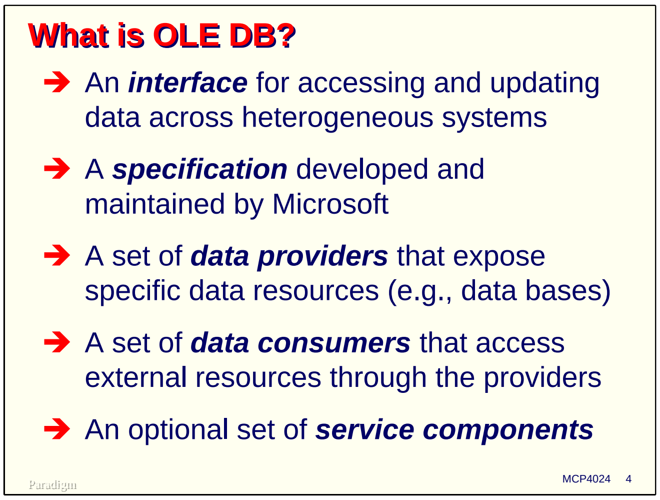
What are we trying to do with OLE DB and why should we use it?

The short answer is that we are trying to access DMSII data bases from *outside* the MCP environment—typically from a Microsoft Windows environment. We want to enable external processes to directly access DMSII for both retrieval and update, and to do this in a way that maintains all of the integrity and recoverability aspects of DMSII for which it is valued.

In order to use OLE DB, you need to know at least something about several things. A general familiarity with DMSII is necessary, along with some basic data base administrator skills. You will also need standard Windows user skills.

Depending on how you plan to use OLE DB, it may be desirable to have some knowledge of SQL, especially the variant used with Microsoft SQL Server. It may also be helpful to have some knowledge of programming languages and environments under Windows—C++, Visual Basic, or one of the scripting languages such as VBScript. Unless you are programming in C++, you will also want to be familiar with an object-oriented interface to OLE DB, ActiveX Data Objects, or ADO.

In most cases, you will need to know relatively little about OLE DB itself.



What is OLE DB?

- An ***interface*** for accessing and updating data across heterogeneous systems
- A ***specification*** developed and maintained by Microsoft
- A set of ***data providers*** that expose specific data resources (e.g., data bases)
- A set of ***data consumers*** that access external resources through the providers
- An optional set of ***service components***

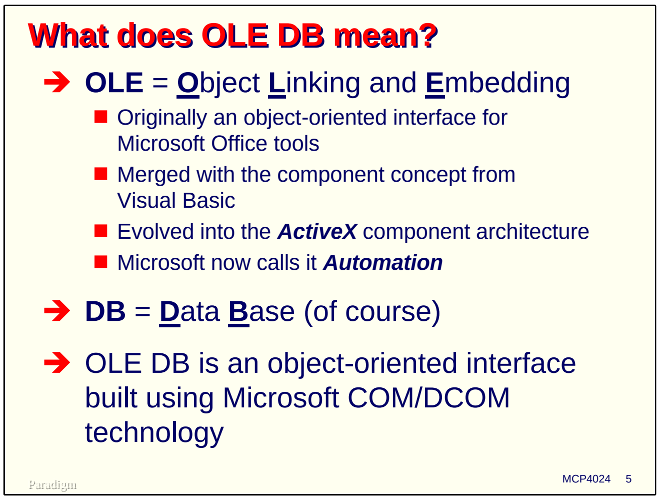
Paradigm MCP4024 4

What is OLE DB? In essence, it is an interface through which you can access and update data sources across heterogeneous systems. It is an open standard, based on a specification that was developed and is maintained by Microsoft. OLE DB is part of the Microsoft Universal Data Access Model, a platform for multi-tier applications designed to operate with diverse data sources.

OLE DB is not just for data bases. It can also be used with file systems, mail stores, and other sources of data. The only requirement is that the provider be able to represent its data in a rectangular form—rows and columns.

OLE DB is also a body of software. It comprises a set of data providers which expose data sources and a set of data consumers which communicate with the providers to retrieve and update data.

Finally, OLE DB comprises a set of optional service components which perform value-added services for both the providers and consumers.



What does OLE DB mean?

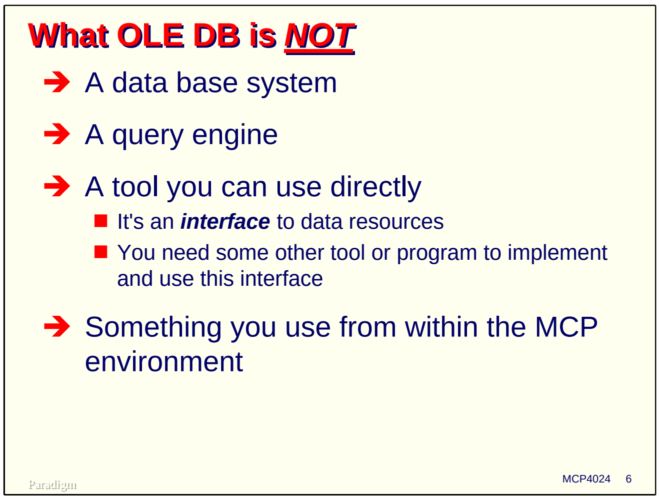
- **OLE** = **O**bject **L**inking and **E**mbedding
  - Originally an object-oriented interface for Microsoft Office tools
  - Merged with the component concept from Visual Basic
  - Evolved into the ***ActiveX*** component architecture
  - Microsoft now calls it ***Automation***
- **DB** = **D**ata **B**ase (of course)
- OLE DB is an object-oriented interface built using Microsoft COM/DCOM technology

Paradigm MCP4024 5

The name "OLE DB" comes from two parts. The first, OLE, stands for Object Linking and Embedding, a Microsoft technology originally developed as an object-oriented interface for Microsoft Office tools. This technology was eventually merged with the component-based development model in Visual Basic, and evolved into what is commonly known as the ActiveX component architecture. Microsoft marketing now refers to ActiveX as "Automation."

DB, obviously, stands for Data Base.

Therefore, OLE DB is an object-oriented interface for data bases, implemented as a set of Microsoft Component Object Model (COM/DCOM) interfaces.



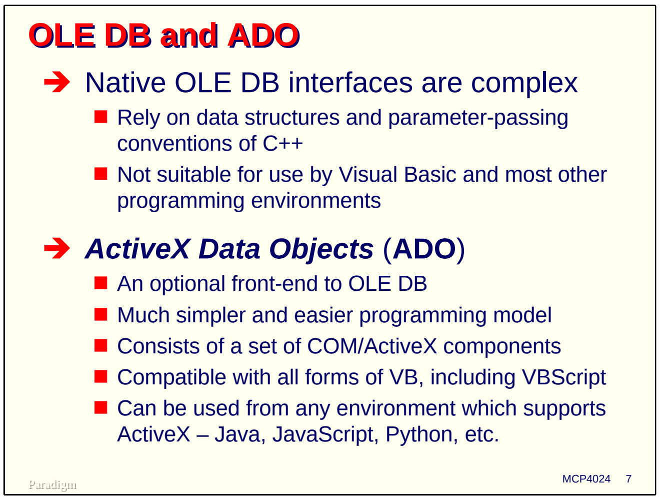
What OLE DB is *NOT*

- A data base system
- A query engine
- A tool you can use directly
  - It's an ***interface*** to data resources
  - You need some other tool or program to implement and use this interface
- Something you use from within the MCP environment

Paradigm MCP4024 6

Having talked about what OLE DB is, we should also discuss what it is *not*. It is not, itself, a data base system, a query engine, or a tool that you can use directly. It is instead an *interface* to data resources. You will need a tool or some sort of programming facility to exercise this interface to actually operate on the data source.

OLE DB is also not something you generally use from within the MCP environment. A portion of OLE DB runs within the MCP environment, but you communicate with that portion from a client outside the system.



OLE DB and ADO

- Native OLE DB interfaces are complex
  - Rely on data structures and parameter-passing conventions of C++
  - Not suitable for use by Visual Basic and most other programming environments
- ***ActiveX Data Objects*** (ADO)
  - An optional front-end to OLE DB
  - Much simpler and easier programming model
  - Consists of a set of COM/ActiveX components
  - Compatible with all forms of VB, including VBScript
  - Can be used from any environment which supports ActiveX – Java, JavaScript, Python, etc.

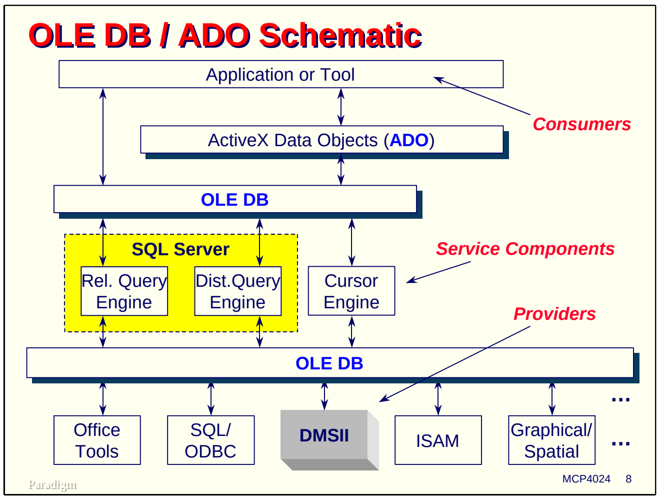
Paradigm MCP4024 7

There is a companion interface with OLE DB that properly should be regarded as a part of it.

The native interfaces and programming model for OLE DB are complex, and are based on data structures and parameter-passing conventions of C++. Because of this, OLE DB cannot be used directly by most other programming languages, such as Visual Basic.

Since a large amount of application development takes place in languages other than C++, particularly Visual Basic and VBScript, Microsoft has developed another COM/ActiveX interface that acts as a front end to OLE DB. This front end is known as ActiveX Data Objects, or ADO.

ADO has much simpler programming model than native OLE DB, and is much easier to program for. It is compatible with all forms of Visual Basic, including VBA (the version in the Office suite) and VBScript. In fact, ADO can be used from any environment which supports ActiveX components, including Java, JavaScript, ECMAScript, Python, etc.



This diagram illustrates the structure of OLE DB and ADO, showing how the various components relate to each other.

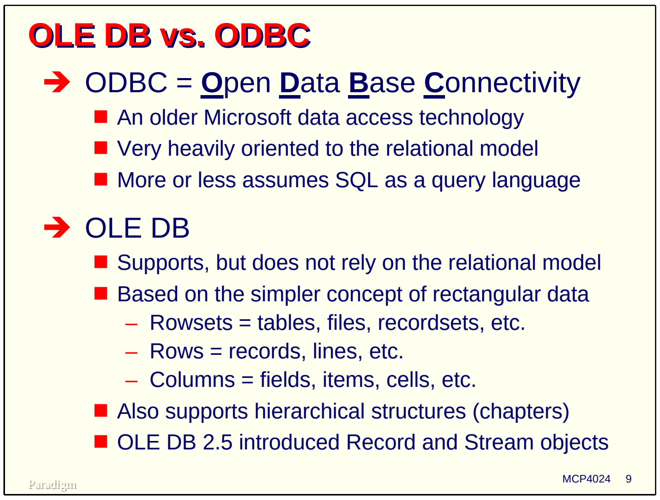
At the top are data consumers, typically applications or tools. These could be purchased software packages (such as Crystal Reports) or applications you write in C++, Visual Basic, or some other language. Along the bottom are OLE DB providers that expose various types of data sources. One of those is the OLE DB Provider for DMSII.

OLE DB itself consists of two pieces that sit between the consumers and providers, along with the default service component that comes with OLE DB, called the Cursor Engine.

If you are using a language other than C++, you will probably be using ADO, which acts as an interface between the consumer and OLE DB.

There are some optional service components which can be included in this arrangement. Relational query engines translate relational data base queries (e.g., SQL) into more primitive OLE DB operations, which are then passed along to the appropriate providers. A distributed query engine is responsible for assembling results of a query that uses data from more than one data source (typically residing on more than one host system).

Microsoft SQL Server can provide both relational and distributed query capabilities, and can be used as a service provider with OLE DB. Interestingly, SQL Server can take on all of the roles in the OLE DB architecture—consumer, provider, and service component. I will talk more later about using SQL Server with the OLE DB Provider for DMSII.



OLE DB vs. ODBC

- ODBC = **O**pen **D**ata **B**ase **C**onnectivity
  - An older Microsoft data access technology
  - Very heavily oriented to the relational model
  - More or less assumes SQL as a query language
- OLE DB
  - Supports, but does not rely on the relational model
  - Based on the simpler concept of rectangular data
    - Rowsets = tables, files, recordsets, etc.
    - Rows = records, lines, etc.
    - Columns = fields, items, cells, etc.
  - Also supports hierarchical structures (chapters)
  - OLE DB 2.5 introduced Record and Stream objects

Paradigm MCP4024 9

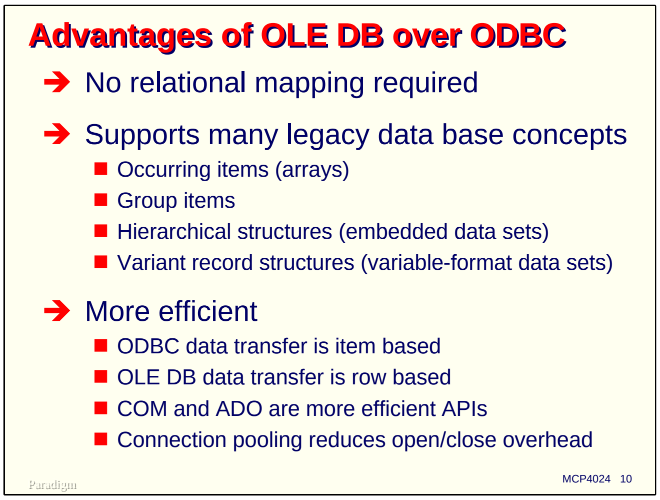
OLE DB tends to be used for many of the same things that ODBC does, so it's useful to compare the two.

ODBC, which stands for Open Data Base Connectivity, is an older Microsoft data access technology. It was originally designed as a way to connect desktop applications to mainframe and large server data bases. ODBC relies very heavily on the relational data model, and more or less assumes that queries will be posed in some variant of SQL.

In contrast, OLE DB supports, but does not rely on the relational model. OLE DB is based on a much simpler model, *rectangular* data. OLE DB deals with data primarily as *rowsets*, which can be entire tables or files, or the results of queries, such as recordsets. A rowset consists simply of rows and columns, with one data element in the cell at each intersection of a row and column.

OLE DB also supports hierarchical data structures, which it terms *chapters*. In essence, a cell in a rowset can itself be a rowset.

OLE DB version 2.5 introduced two new types of objects, Record and Stream, which are useful for dealing with non-traditional data stores, such as email repositories and file systems with hierarchical folder structures.



Advantages of OLE DB over ODBC

- No relational mapping required
- Supports many legacy data base concepts
  - Occurring items (arrays)
  - Group items
  - Hierarchical structures (embedded data sets)
  - Variant record structures (variable-format data sets)
- More efficient
  - ODBC data transfer is item based
  - OLE DB data transfer is row based
  - COM and ADO are more efficient APIs
  - Connection pooling reduces open/close overhead

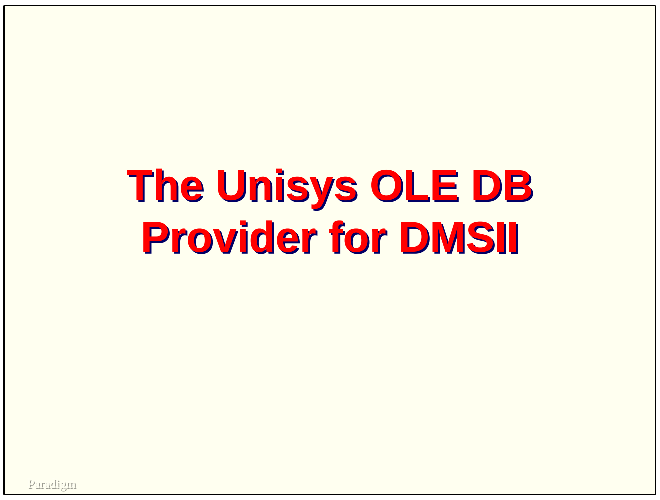
Paradigm MCP4024 10

Microsoft learned a number of lessons from their experience with ODBC and applied those to its design of OLE DB. As a result, OLE DB has a number of advantages over ODBC.

One of the major advantages, especially when dealing with non-relational data sources such as DMSII, is that no relational mapping is required. Data providers have the much simpler job of casting their data into merely a rectangular format.

Due to its simpler data model and hierarchical capabilities, OLE DB can support many legacy data base structure concepts, such as occurring items, group items, embedded data sets, and variable-format records. These types of legacy structures generally cause a good deal of grief when the relational model must be contorted around them.

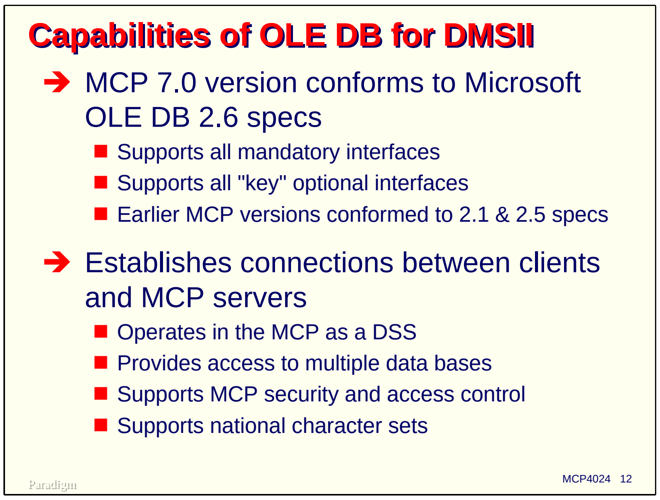
OLE DB is also generally more efficient than ODBC. Part of the reason for this is that data transfer from the data source to the consumer is done on an item-by-item basis with ODBC, while is is done on a row (record) basis with OLE DB. COM and ADO are efficient application programming interfaces, and this also aids the overall efficiency of OLE DB. Finally, OLE DB supports connection pooling, which can dramatically reduce the substantial overhead usually involved in connecting clients to, and disconnecting them from, data sources.



The Unisys OLE DB Provider for DMSII

Paradigm

With that brief overview of OLE DB in general, let us now take a look at the specific capabilities and features of the Unisys OLE DB Provider for DMSII.



Capabilities of OLE DB for DMSII

- MCP 7.0 version conforms to Microsoft OLE DB 2.6 specs
  - Supports all mandatory interfaces
  - Supports all "key" optional interfaces
  - Earlier MCP versions conformed to 2.1 & 2.5 specs
- Establishes connections between clients and MCP servers
  - Operates in the MCP as a DSS
  - Provides access to multiple data bases
  - Supports MCP security and access control
  - Supports national character sets

Paradigm MCP4024 12

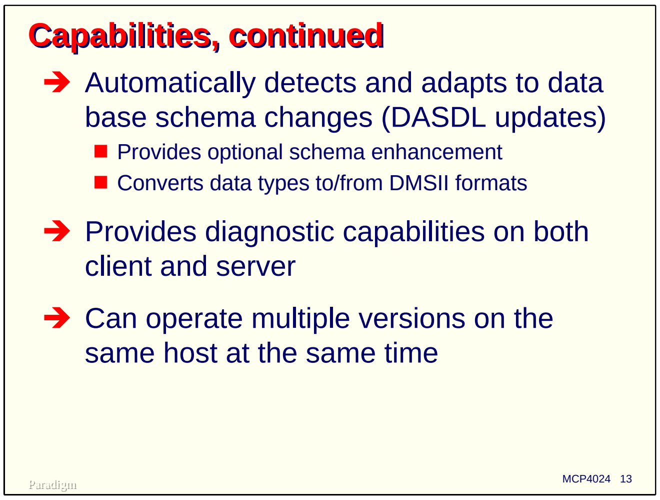
The MCP 7.0 (SSR 48.1) version of OLE DB conforms to the Microsoft version 2.6 specification. Earlier versions, starting with MCP 5.0, conformed with the 2.1 version of the specification and operated well with version 2.5 components.

The Unisys Provider supports all mandatory OLE DB interfaces and all of the optional interfaces which Microsoft has identified as key ones.

One of the main functions of the OLE DB Provider is to establish connections between clients and MCP servers. One part of the Provider resides on the client; the other part resides in the MCP environment and operates as a Distributed Systems Service (DSS). This means you can communicate with the MCP portion of the Provider for purposes of control and administration using NA commands from the ODT.

A single installation of OLE DB can support any number of data bases simultaneously. The server library fires off worker stacks as necessary to perform data retrieval and update functions.

OLE DB fully supports MCP security and access control, including guardfiles, accesscodes, and access passwords. It also supports national character sets for systems operating in non-English environments.



Capabilities, continued

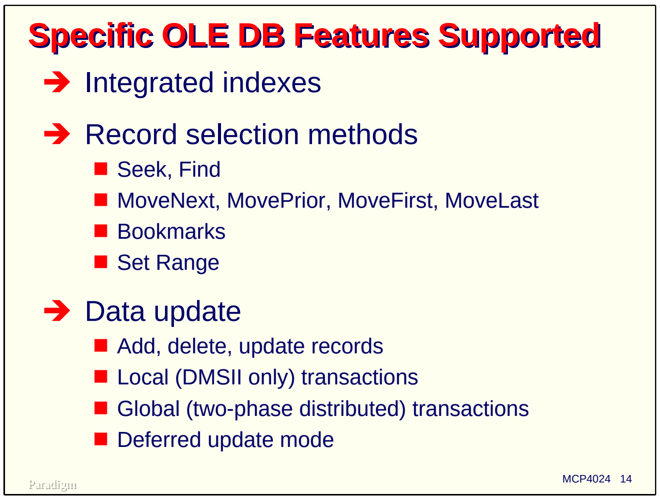
- Automatically detects and adapts to data base schema changes (DASDL updates)
  - Provides optional schema enhancement
  - Converts data types to/from DMSII formats
- Provides diagnostic capabilities on both client and server
- Can operate multiple versions on the same host at the same time

Paradigm MCP4024 13

One of the nicest features of OLE DB is that it automatically generates the schema information it needs internally, and automatically detects when a DASDL update has modified the data base schema. It provides a utility program that allows you to enhance the schema by changing data names and overriding default DMSII data types. The Provider converts native DMSII data formats to the formats required by OLE DB, ensuring that data will be translated correctly between heterogeneous environments.

The Provider provides diagnostic capabilities, in the form of dumps and traces, in both the MCP and Windows environments.

You can operate multiple versions of the Provider on the same host at the same time. This is especially useful for sites which need to run multiple versions of DMSII on their systems.



Specific OLE DB Features Supported

- Integrated indexes
- Record selection methods
  - Seek, Find
  - MoveNext, MovePrior, MoveFirst, MoveLast
  - Bookmarks
  - Set Range
- Data update
  - Add, delete, update records
  - Local (DMSII only) transactions
  - Global (two-phase distributed) transactions
  - Deferred update mode

Paradigm MCP4024 14

The Unisys Provider supports a number of specific OLE DB features.

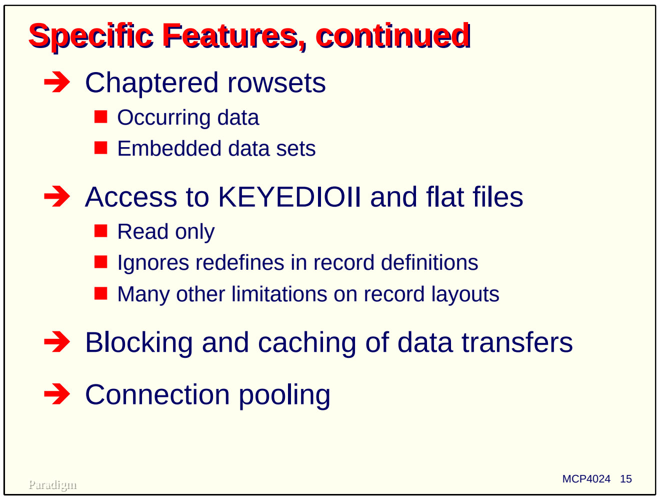
Integrated indexes allow OLE DB (and some data consumers) to recognize DMSII indexes as an integral part of their data sets and to pass higher-level retrieval requests through to DMSII. This allows DMSII to perform random retrievals using the indexes. Without this capability, consumers would be reduced to reading data sets sequentially.

The Provider supports a number of record selection and retrieval methods:

- Seek and Find methods allow random retrieval of records based on keys, and some limited forms of record filtering.
- The various Move methods allow sequential access in either a forward or reverse direction through a data set or an index.
- Bookmarks allow you to retrieve an object which uniquely describes the location of a record in a data set, and later retrieve that record efficiently. Bookmarks are also used for DMSII links.
- Set Range establishes key limits for retrieving a range of records via an integrated index. It is not available in ADO, but is used extensively by SQL Server.

The Provider also supports a complete set of methods for updating DMSII data. These include:

- Adding, deleting, and inserting records in data sets.
- Local transactions that correspond to DMSII Begin- and End-Transaction statements. If the consumer does not explicitly enter and exit transaction state, the Provider implicitly creates a transaction around each update method call.
- Global, or two-phase distributed transactions. These allow DMSII data bases to participate in transactions which span multiple data sources, typically on separate, heterogeneous hosts.
- Deferred update mode. ADO has the concept of server-side and client-side cursors. With a client-side cursor, you effectively download a set of data to your client, update it locally, and then save the updated result back to the data source. ADO determines the differences between the original and updated values, and OLE DB creates update transactions to post those updates to the data source.



Specific Features, continued

- Chaptered rowsets
  - Occurring data
  - Embedded data sets
- Access to KEYEDIOII and flat files
  - Read only
  - Ignores redefines in record definitions
  - Many other limitations on record layouts
- Blocking and caching of data transfers
- Connection pooling

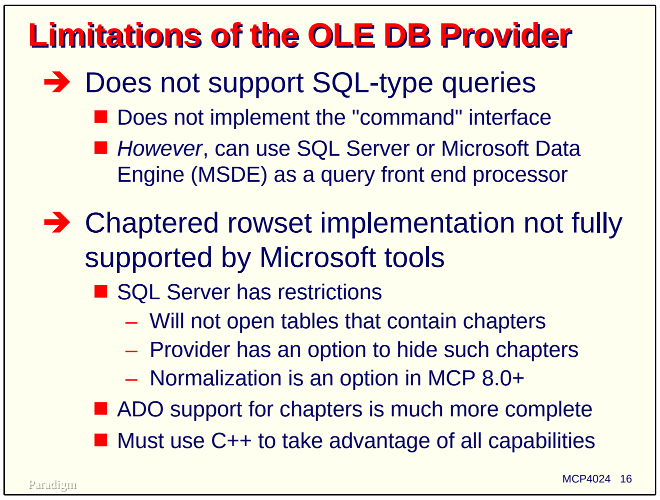
Paradigm MCP4024 15

Chaptered rowsets are the vehicle by which OLE DB can represent and act upon DMSII occurring items and embedded data sets. A chaptered rowset is effectively a rowset nested within another rowset. The concept is very similar to that of an DMSII embedded data set. There are other ways of dealing with these legacy structures in addition to chapters. Occurring items can be "unrolled" so that they appear to OLE DB as separate scalar values. With MCP 8.0, embedded data sets can be "normalized," which makes them appear to OLE DB as a join of the embedded data with its master record.

The OLE DB Provider also supports access to KEYEDIOII files and flat files, but only in a read-only manner. There are a number of limitations to the record layouts for these types of files that can be used by the Provider (e.g., redefines are ignored), so unless the file is specifically designed for use with the Provider, you may find this capability of limited utility. I will not discuss the KEYEDIOII and flat file capability further in this presentation, but rather focus on using OLE DB with DMSII data bases.

The Provider also supports blocking and caching of data transfers. In ADO, you can use the CacheSize property of recordsets to specify that rows of the recordset should be blocked and cached during data transfer. This serves to improve the overall data transfer rate between the data source and the consumer, especially when retrieving large amounts of data from a single request.

Finally, the Provider for DMSII supports connection pooling, which can reduce the amount of time required to open and close connections between the client and server systems.



Limitations of the OLE DB Provider

- Does not support SQL-type queries
  - Does not implement the "command" interface
  - *However*, can use SQL Server or Microsoft Data Engine (MSDE) as a query front end processor
- Chaptered rowset implementation not fully supported by Microsoft tools
  - SQL Server has restrictions
    - Will not open tables that contain chapters
    - Provider has an option to hide such chapters
    - Normalization is an option in MCP 8.0+
  - ADO support for chapters is much more complete
  - Must use C++ to take advantage of all capabilities

Paradigm MCP4024 16

Along with its features, the OLE DB Provider for DMSII has some limitations.

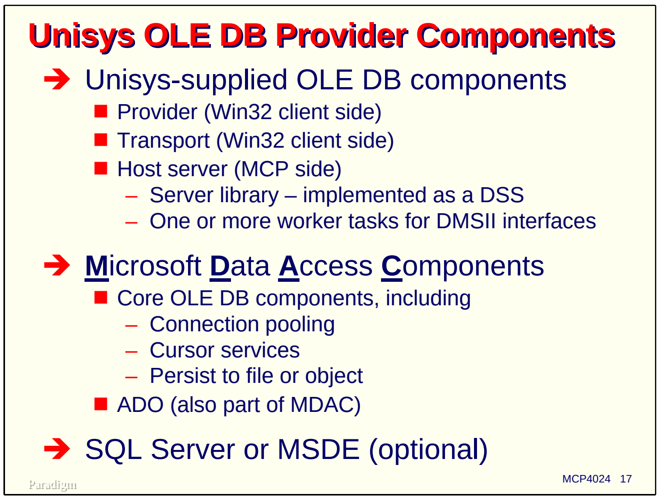
One of the major limitations is that it does not support SQL-type queries directly. The OLE DB specifications provide for a "command" interface, but the DMSII Provider does not currently implement this.

You can, however, use SQL Server or Microsoft Data Engine (MSDE, a stripped-down version of SQL Server designed as a replacement for Jet in desktop environments) as a front-end processor to the OLE DB Provider. You feed SQL queries to SQL Server, which breaks them down into more primitive operations, and passes them via OLE DB to DMSII for execution. This is a very nice capability, which I will discuss further later in the presentation.

Another major limitation is that some Microsoft tools do not support the chaptered rowset implementation used by the Unisys Provider. SQL Server appears to have the most restrictions, as it will not open tables which contain chapters. The Provider has a configuration option that will cause it to hide chapters in the schema, which will then allow SQL Server to open tables that contain chapters, but not access the chaptered data. Occurring items can be unrolled, and in MCP 8.0, embedded data sets can be presented to SQL Server as normalized structures, although they are read only when presented this way. Unisys has announced an update capability for normalized structures in MCP 9.0.

ADO support for chapters is now much more complete. This did not work with earlier versions of the Provider, but at least as of MCP 7.0 and ADO 2.6, chapters appear to work in ADO.

To take advantage of all the capabilities of chapters, you must use C++ and the native OLE DB interface. For most of us, fortunately, this is not necessary.



Unisys OLE DB Provider Components

- Unisys-supplied OLE DB components
  - Provider (Win32 client side)
  - Transport (Win32 client side)
  - Host server (MCP side)
    - Server library – implemented as a DSS
    - One or more worker tasks for DMSII interfaces
- Microsoft Data Access Components
  - Core OLE DB components, including
    - Connection pooling
    - Cursor services
    - Persist to file or object
  - ADO (also part of MDAC)
- SQL Server or MSDE (optional)

Paradigm MCP4024 17

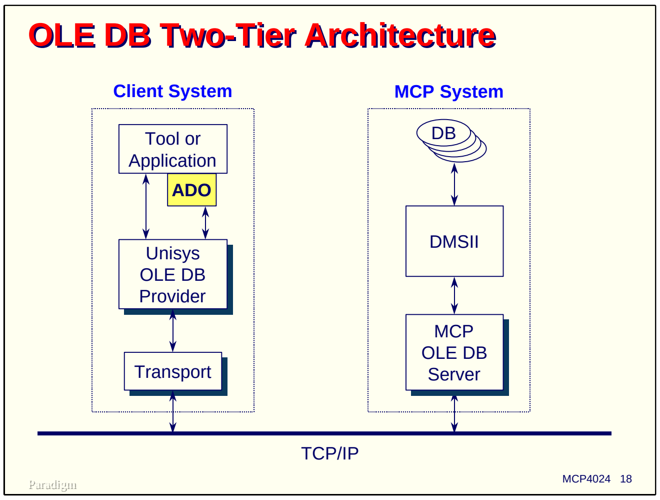
To install and use OLE DB you need at least two sets of components, and possibly three.

The Unisys Provider comes in three parts:

- The Provider itself, which is installed on and operates in a Windows environment. This could be a Windows server, or a Windows desktop, including Windows 9x.
- The Transport, which handles communications between the Windows consumer and the MCP data source. This also operates in the Windows environment.
- The host Server, which is installed on and operates in the MCP environment. The Server is implemented in two pieces, a DSS library, and a worker module, multiple copies of which are fired off by the library to process individual requests.

The second thing you need is the core Windows software for OLE DB. This is contained in a Microsoft module called the Microsoft Data Access Components, or MDAC (pronounced EM-dak). In addition to the core OLE DB elements, MDAC contains ADO and several Microsoft providers, including ones for SQL Server, Jet, and a bridge that allows OLE DB to work with any ODBC driver.

If you want to process SQL queries against DMSII data, you need some form of Microsoft SQL Server or MSDE. This component is optional, and you can do a lot with the OLE DB Provider without it.

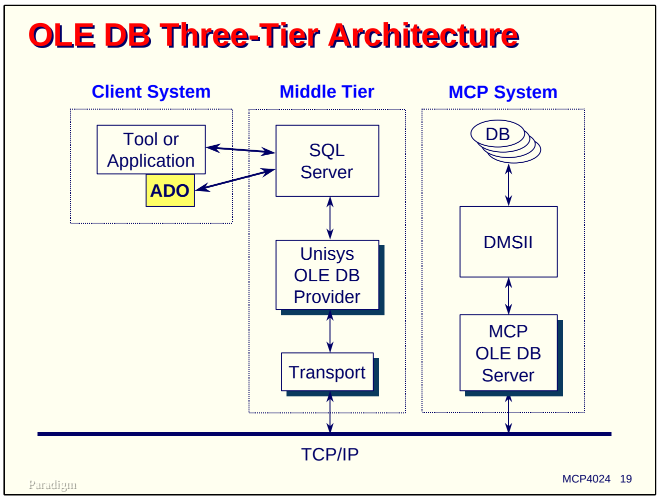


OLE DB Two-Tier Architecture

This diagram shows how the OLE DB Provider can be used in a two-tier environment. On the right is the MCP system, with its DMSII data bases and the Server portion of the Provider. On the left is a client system with the two Windows-based portions of the Provider, the tool or application acting as a data consumer, and optionally, ADO.

The core OLE DB components pass requests to the Windows-resident portion of the Provider, from which they are sent over a standard TCP/IP connection (port 1871 by default) to the MCP host. The Server portion of the Provider on the MCP host translates the requests to primitive DMSII operations, and passes them to the Accessroutines for execution. The results flow back across the TCP/IP connection to the Windows system and are returned to the originating application.

When used in this manner, the data consumer is restricted essentially to record-level operations, much as you would do in an MCP COBOL program.

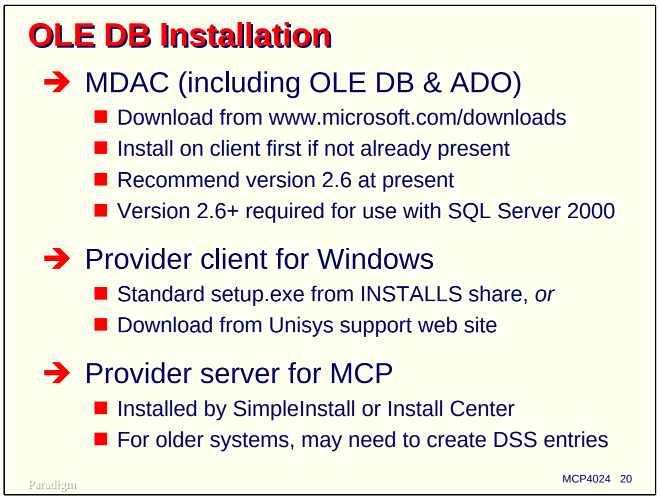


OLE DB Three-Tier Architecture

This next diagram illustrates OLE DB in a three-tier environment. The arrangement on the MCP host at the right is the same as for a two-tier arrangement. The middle tier, however, uses SQL Server as a query front-end processor, which in turn talks to the Windows-resident Provider components. The client system sends SQL requests to SQL Server, which breaks them down into more primitive OLE DB requests before passing them to the Provider.

Note that in this case the "client" from the OLE DB Provider's perspective is SQL Server, not the end user.

With this three-tier arrangement, it is very easy to perform distributed queries across multiple data sources on different types of systems, and to do two-phase distributed update transactions. I will discuss this topic in more detail later in the presentation.



OLE DB Installation

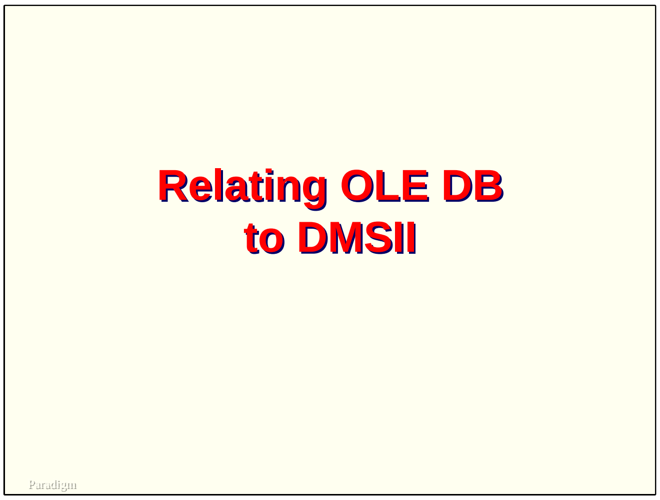
- MDAC (including OLE DB & ADO)
  - Download from www.microsoft.com/downloads
  - Install on client first if not already present
  - Recommend version 2.6 at present
  - Version 2.6+ required for use with SQL Server 2000
- Provider client for Windows
  - Standard setup.exe from INSTALLS share, *or*
  - Download from Unisys support web site
- Provider server for MCP
  - Installed by SimpleInstall or Install Center
  - For older systems, may need to create DSS entries

Paradigm MCP4024 20

To install OLE DB in your environment, you need both a Windows system and an MCP system. Start by installing MDAC on the Windows system, if it is not already there. Unless you have a specific reason to do otherwise, I recommend that you install MDAC version 2.6. If you are going to be using SQL Server 2000 with OLE DB, you need at least version 2.6. You can download MDAC for free from the Microsoft web site.

Next, install the Windows-based components of the Unisys Provider. You must have MDAC on the Windows system before you install the Unisys Provider. These Windows-based components can be found in the Installs share on ClearPath MCP systems, or you can download them from the Unisys support site.

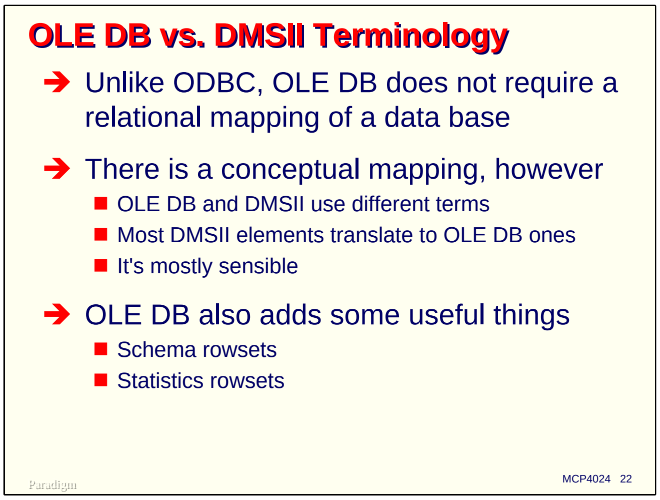
You also need to install the MCP-based components. Normally this will be done for you automatically by SimpleInstall or Install Center. If you have an older system that was delivered with pre-MCP 5.0 software, you may need to manually create the necessary DSS configuration. The OLE DB *Operations Guide* has detailed instructions on how to do this.



Relating OLE DB to DMSII

Paradigm

In this next section I will discuss several OLE DB concepts and how they relate to DMSII concepts.



OLE DB vs. DMSII Terminology

- Unlike ODBC, OLE DB does not require a relational mapping of a data base
- There is a conceptual mapping, however
  - OLE DB and DMSII use different terms
  - Most DMSII elements translate to OLE DB ones
  - It's mostly sensible
- OLE DB also adds some useful things
  - Schema rowsets
  - Statistics rowsets

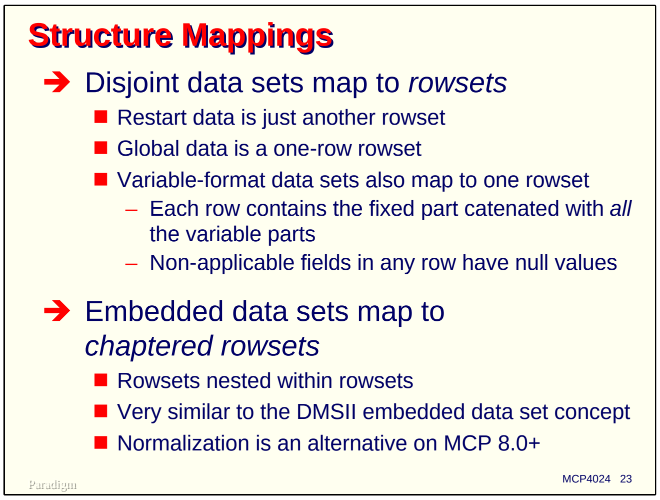
Paradigm MCP4024 22

With ODBC products for DMSII, you must map DMSII entities to relational counterparts. With OLE DB this relational mapping is not necessary.

There is, however, a conceptual mapping between OLE DB and DMSII. OLE DB uses different terms than we are used to with DMSII. Most DMSII elements translate directly to OLE DB ones. In most cases this translation is direct and sensible.

The OLE DB Provider also adds a couple of useful things that we do not normally have in the native DMSII environment:

- Schema rowsets allow you to obtain information on the structure of a DMSII data base, including the tables, data items within those tables, and indexes to the tables. This is very useful for generic tools which can adapt themselves to different data bases by "discovering" their structure through these schema rowsets.
- Statistics rowsets. The Provider maintains table-by-table usage statistics for a connection. You can read these statistics through the statistics rowsets.



Structure Mappings

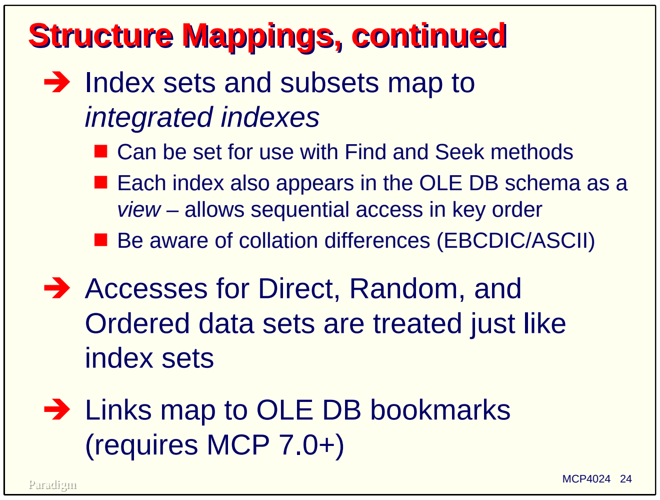
- Disjoint data sets map to *rowsets*
  - Restart data is just another rowset
  - Global data is a one-row rowset
  - Variable-format data sets also map to one rowset
    - Each row contains the fixed part catenated with *all* the variable parts
    - Non-applicable fields in any row have null values
- Embedded data sets map to *chaptered rowsets*
  - Rowsets nested within rowsets
  - Very similar to the DMSII embedded data set concept
  - Normalization is an alternative on MCP 8.0+

Paradigm MCP4024 23

Disjoint DMSII data sets map to OLE DB rowsets. Restart data sets are treated like any other data set. Global data, if present in the DMSII data base, is treated like a one-row data set.

Variable-format data sets are represented in a much more sensible way by the OLE DB Provider than they are by most ODBC implementations. Each row in the rowset for a variable-format data set contains the fields of the fixed (or head) part of the record catenated with the fields from *all* of the variable (or tail) parts. In each of these rows, the fields for the head and at most one of the tails will be valid. Fields for all of the other tail parts will automatically be provided with NULL values.

Embedded data sets map directly to chaptered rowsets. As mentioned earlier, these are rowsets nested within rowsets. The concept and the representation are so similar to that for DMSII embedded data sets it almost seems like chapters were designed explicitly to support DMSII. In the MCP 8.0 version of the Provider, you can also have embedded data sets represented as normalized structures, which allows them to be accessed by SQL Server.



Structure Mappings, continued

- Index sets and subsets map to *integrated indexes*
  - Can be set for use with Find and Seek methods
  - Each index also appears in the OLE DB schema as a *view* – allows sequential access in key order
  - Be aware of collation differences (EBCDIC/ASCII)
- Accesses for Direct, Random, and Ordered data sets are treated just like index sets
- Links map to OLE DB bookmarks (requires MCP 7.0+)

Paradigm MCP4024 24

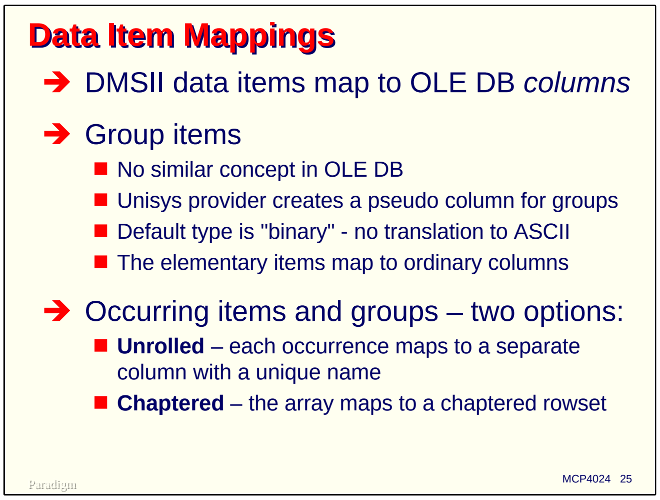
Index sets and subsets map to OLE DB integrated indexes. These indexes can be specified for use with Find and Seek methods in ADO.

Each index also appears as a *view* in the OLE DB schema. A view has many of the same characteristics as its data set. You can open one of these views and read its records sequentially, which will cause them to be returned in the key order of that index.

You need to be aware of collation differences when using indexes with OLE DB. Most Windows applications use ASCII, ANSI, or Unicode internally. The MCP, of course, uses EBCDIC internally, so records returned from DMSII data sources can come in a different order than equivalent records from other data sources.

If your data base has Direct, Random, or Ordered data sets, the accesses for those data sets are treated just like index sets.

Beginning with MCP 7.0, the OLE DB Provider supports DMSII links. These appear as Bookmark objects to the data consumer.



Data Item Mappings

- DMSII data items map to OLE DB *columns*
- Group items
  - No similar concept in OLE DB
  - Unisys provider creates a pseudo column for groups
  - Default type is "binary" - no translation to ASCII
  - The elementary items map to ordinary columns
- Occurring items and groups – two options:
  - **Unrolled** – each occurrence maps to a separate column with a unique name
  - **Chaptered** – the array maps to a chaptered rowset

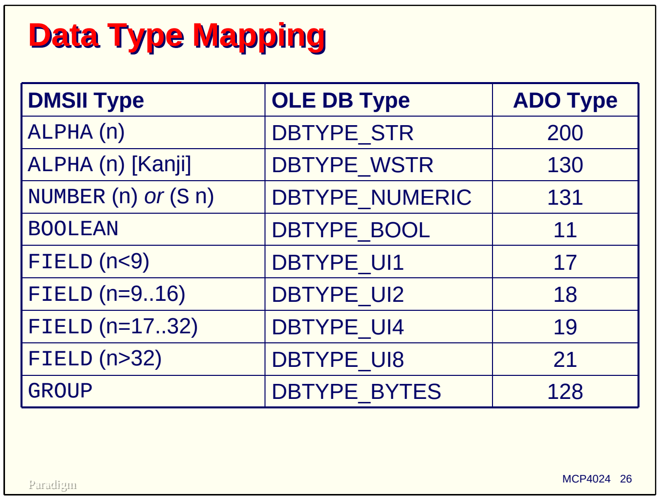
Paradigm MCP4024 25

DMSII data items map to OLE DB columns. There is a reasonable mapping for each of the elementary DMSII data types. The next two slides show specifically how items are mapped.

OLE DB does not have the concept of group items. To compensate for this, the OLE DB Provider maps group items as a separate pseudo column. Each of the elementary items for the group is also mapped into a column of appropriate type. The default data type for the group pseudo column is "binary"—no translation. You can override this using the Schema Utility.

With occurring items you have two options:

- You can "unroll" the occurring items using the Schema Utility. This makes the occurring items appear to the data consumer as a set of individual data items, each with its own name.
- The occurring items can be represented as a chapter. This is the default, and as mentioned earlier, cannot be used if you plan to access the data through SQL Server.

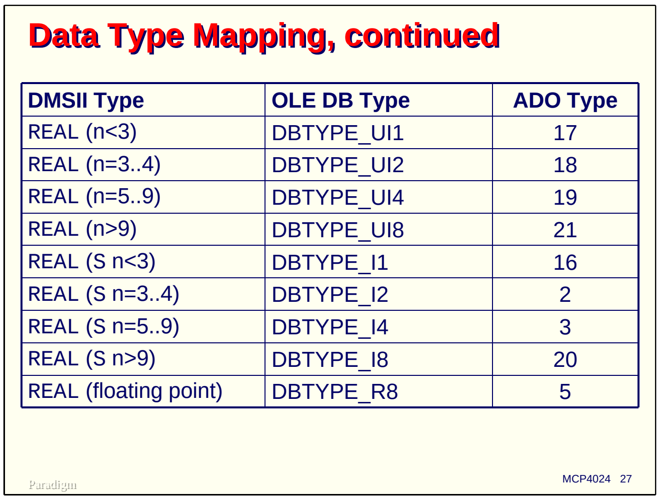


Data Type Mapping

DMSII Type	OLE DB Type	ADO Type
ALPHA (n)	DBTYPE_STR	200
ALPHA (n) [Kanji]	DBTYPE_WSTR	130
NUMBER (n) *or* (S n)	DBTYPE_NUMERIC	131
BOOLEAN	DBTYPE_BOOL	11
FIELD (n<9)	DBTYPE_UI1	17
FIELD (n=9..16)	DBTYPE_UI2	18
FIELD (n=17..32)	DBTYPE_UI4	19
FIELD (n>32)	DBTYPE_UI8	21
GROUP	DBTYPE_BYTES	128

Paradigm MCP4024 26

This chart simply shows how DMSII data types are mapped into OLE DB and ADO types.

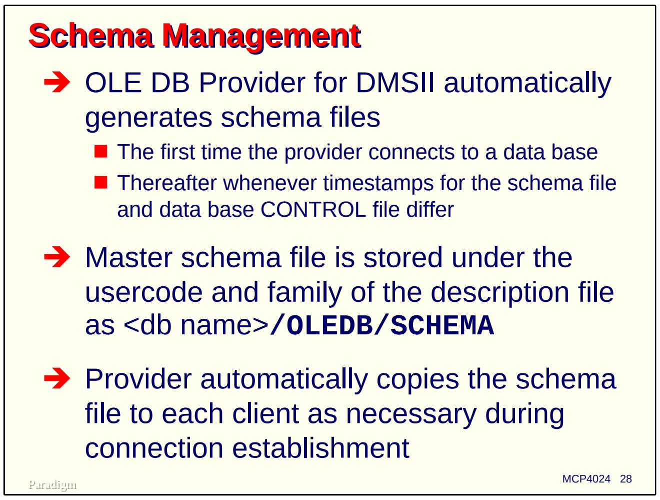


Data Type Mapping, continued

DMSII Type	OLE DB Type	ADO Type
REAL (n<3)	DBTYPE_UI1	17
REAL (n=3..4)	DBTYPE_UI2	18
REAL (n=5..9)	DBTYPE_UI4	19
REAL (n>9)	DBTYPE_UI8	21
REAL (S n<3)	DBTYPE_I1	16
REAL (S n=3..4)	DBTYPE_I2	2
REAL (S n=5..9)	DBTYPE_I4	3
REAL (S n>9)	DBTYPE_I8	20
REAL (floating point)	DBTYPE_R8	5

Paradigm MCP4024 27

DMSII REAL types are mapped in to a variety of OLE DB types, depending on the precision of the number specified in DASDL.



Schema Management

- OLE DB Provider for DMSII automatically generates schema files
  - The first time the provider connects to a data base
  - Thereafter whenever timestamps for the schema file and data base CONTROL file differ
- Master schema file is stored under the usercode and family of the description file as <db name>**`/OLEDB/SCHEMA`**
- Provider automatically copies the schema file to each client as necessary during connection establishment

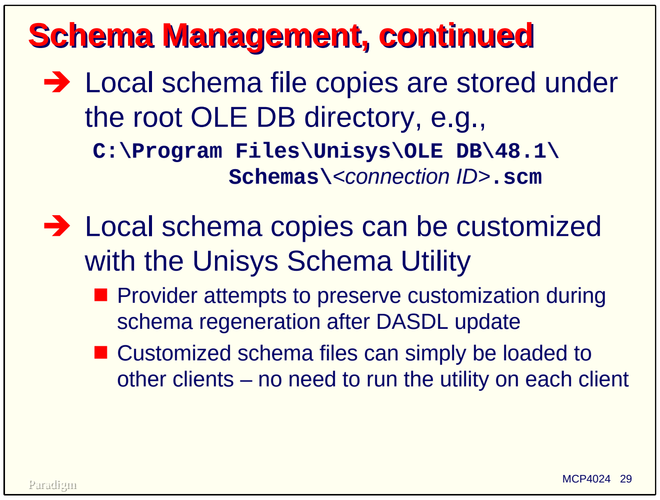
Paradigm MCP4024 28

One of the best features of the OLE DB Provider is how it manages its schema data and how it adapts to changes in the definition of the data base.

The first time you connect to a DMSII data base using OLE DB, the Server on the MCP host generates a file of schema data and stores it under the same user and family as the description file with the title <dbname>**`/OLEDB/SCHEMA`**.

This schema file contains the timestamp of the data base. Thereafter, when you connect to the data base, the Server checks this timestamp against the control file. If it has changed, the Server knows that a DASDL update has occurred, and it automatically regenerates the schema file to obtain the latest definition of the data base. This generation of the schema file usually takes just a few seconds.

A copy of the schema data is also stored on each client system. The Provider automatically copies the schema data to the client when necessary.



Schema Management, continued

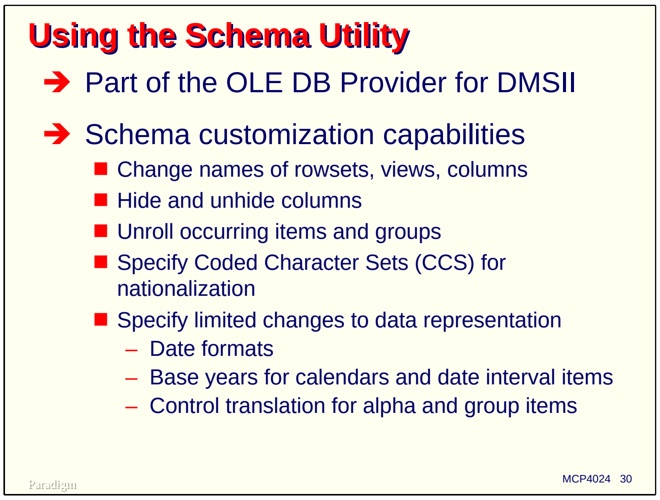
- Local schema file copies are stored under the root OLE DB directory, e.g.,
  `C:\Program Files\Unisys\OLE DB\48.1\`
  `Schemas\`*`<connection ID>`***`.scm`**
- Local schema copies can be customized with the Unisys Schema Utility
  - Provider attempts to preserve customization during schema regeneration after DASDL update
  - Customized schema files can simply be loaded to other clients – no need to run the utility on each client

Paradigm MCP4024 29

The local copies of the schema are stored on the client under the Schemas folder of the OLE DB Provider's installation directory. These local copies can be customized with the Schema Utility, which is installed on the client along with the Provider.

Note that the schema customizations are present only in the local client copies of the schema data. They are not in the master schema file resident on the MCP host. When the schema data is regenerated after a DASDL update, the Provider attempts to preserve as much of the customization as possible.

A master copy of a customized schema can be created on one client and simply copied to other clients. It is not necessary to run the Schema Utility and separately customize the schema data on every client system.



Using the Schema Utility

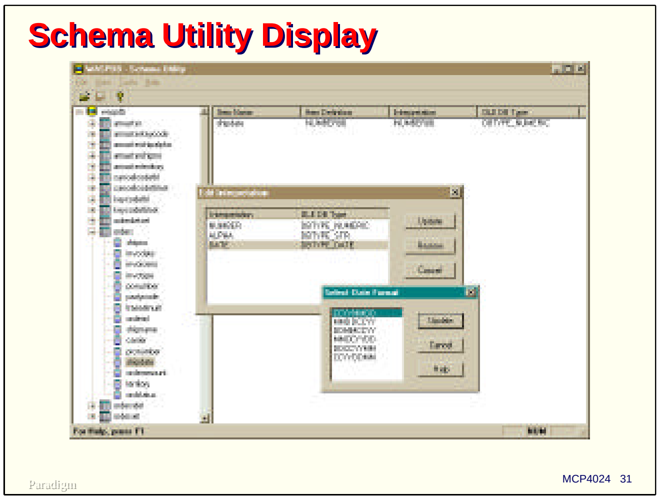
- Part of the OLE DB Provider for DMSII
- Schema customization capabilities
  - Change names of rowsets, views, columns
  - Hide and unhide columns
  - Unroll occurring items and groups
  - Specify Coded Character Sets (CCS) for nationalization
  - Specify limited changes to data representation
    - Date formats
    - Base years for calendars and date interval items
    - Control translation for alpha and group items

Paradigm MCP4024 30

The tool used to customize local schema files is the Schema Utility. This is a Windows GUI which is installed along with the other Windows-based components of the Provider.

The Schema Utility allows you to apply the following types of customizations for a data base:

- Change the names of the rowsets, views, and columns in the schema. By default these have their DASDL names, translated to lower case, with hyphens replaced by underscores.
- Hide columns so they are not visible to OLE DB consumers and unhide previously-hidden columns.
- Unroll and re-roll occurring items and groups. Once unrolled, the individual data items can have their names changed.
- Specify Coded Character Sets (CCS) for nationalization of data representations.
- Specify some limited changes to the way the Provider represents data to consumers.
  - Numeric and alpha items which contain dates can have the date format specified.
  - Numeric items can be represented as alpha and vice versa.
  - Base years for calendars and date interval items can be specified.
  - Translation can be turned on or off. With translation off, data items are passed completely unaltered from DMSII to the consumer. This may be useful if alphanumeric items contain bitmasks or other non-EBCDIC coding.



Schema Utility Display

Item Name	Item Definition	Interpretation	OLE DB Type
shipdate	NUMBER(8)	NUMBER(8)	DBTYPE_NUMERIC

Edit Interpretation

Interpretation	OLE DB Type
NUMBER	DBTYPE_NUMERIC
ALPHA	DBTYPE_STR
DATE	DBTYPE_DATE

Select Date Format

CCYYMMDD
MMDDCCYY
DDMMCCYY
MMDDYYDD
DDCCYYMM
CCYYDDMM

Paradigm

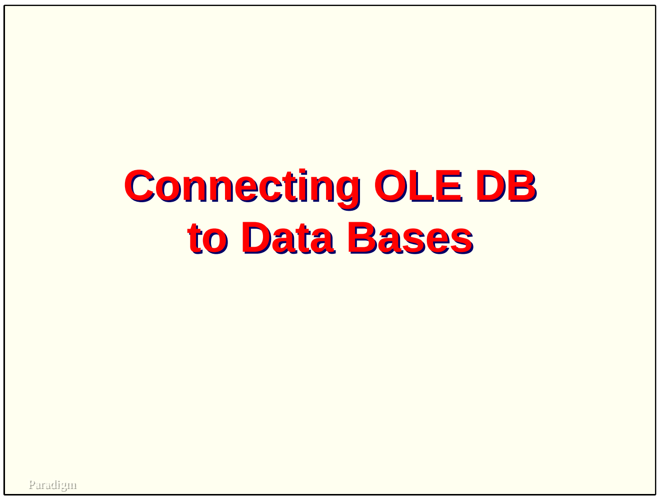
MCP4024 31

This screen shot shows a sample view of the Schema Utility in the process of specifying a date format for a numeric item.

The left panel of the window contains a tree of the items from the data base. You open the root (data base) node to expose data sets, open data sets to expose their data items, and open group items to expose the elementary items within them. Occurring items show the number of occurrences in square brackets after the item name.

Clicking on a name in the left panel causes detailed information for the item to appear in the right panel. Right-clicking on a name in the left panel (double right-clicking in Windows 9x) brings up a context menu from which you can select customization options.

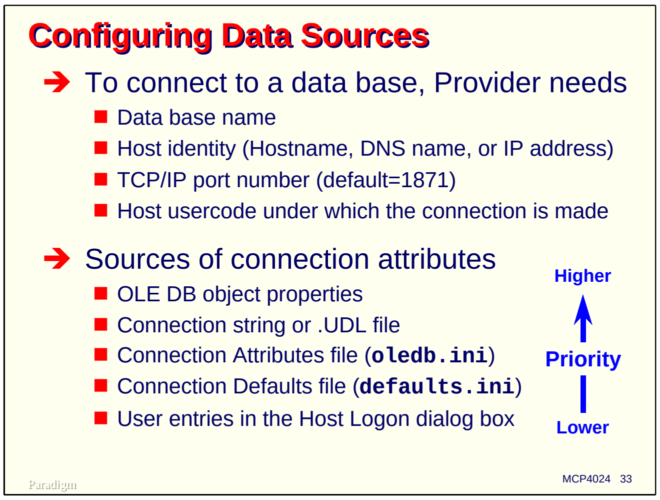
When you are finished customizing a schema, you simply save it and exit the Schema Utility program.



Connecting OLE DB to Data Bases

Paradigm

After installing the OLE DB software but before you can retrieve data from a DMSII data base, you must connect to the data base. This section discusses how those connections are established.



Configuring Data Sources

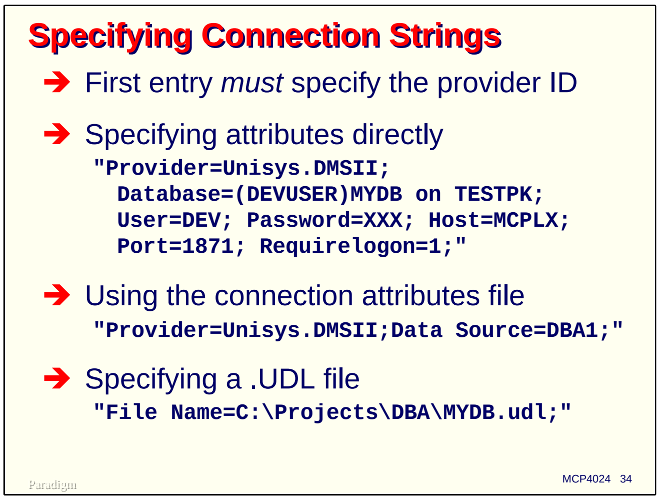
- To connect to a data base, Provider needs
  - Data base name
  - Host identity (Hostname, DNS name, or IP address)
  - TCP/IP port number (default=1871)
  - Host usercode under which the connection is made
- Sources of connection attributes
  - OLE DB object properties
  - Connection string or .UDL file
  - Connection Attributes file (`oledb.ini`)
  - Connection Defaults file (`defaults.ini`)
  - User entries in the Host Logon dialog box

Paradigm MCP4024 33

To connect to a data base, the Windows-resident Provider needs to know four things: the data base name on the MCP host, the host identity (which can be a hostname, DNS name, or IP address), the TCP/IP port number over which the connection will be made (1871 by default), and the usercode and password under which the DMSII data base will be accessed. There are some optional pieces of information that can be provided as well.

These connection attributes can come from a variety of sources, and from a combination of those sources. There is an implicit priority to the attribute values depending on the source they come from. Values for an attribute from sources higher in the list override values for the same attribute from sources lower in the list. The list of sources, in priority order, is:

- OLE DB object properties. These are attribute values that are specified directly to the OLE DB (or ADO) objects. This source is similar to setting file and task attributes at run time in MCP programs.
- The connection string or Universal Data Locator (UDL) file. The Open method for both OLE DB and ADO takes a string parameter. Attributes can be specified in this string using name/value pairs separated by semicolons. UDL files are simply a Microsoft mechanism for storing a connection string in a text file. If you create a file with a `.udl` extension, right click on it in Windows Explorer, and select Properties, Windows will display a tabbed dialog box you can use to fill in the most common attribute values. When you click OK on the dialog box, it stores the resulting connection string in the UDL file. You can look at UDL files with a text editor, but be aware they are stored using Unicode.
- The Unisys OLE DB Provider for DMSII implements another way to store connection attributes in a file, called the Connection Attributes file. I will discuss this file in more detail shortly.
- The Unisys Provider also implements a file of default attributes which will be applied to the connection if they are not overridden at a higher level. I will also discuss this file in more detail shortly.
- Finally, if after going through all of the sources above the Provider still does not have all of the information it needs to establish a connection, it can display a Host Logon dialog box. By default, though, if you have connection pooling enabled (which it is by default), this dialog box will not display and the connection will fail.



Specifying Connection Strings

- First entry *must* specify the provider ID
- Specifying attributes directly

```
"Provider=Unisys.DMSII;
  Database=(DEVUSER)MYDB on TESTPK;
  User=DEV; Password=XXX; Host=MCPLX;
  Port=1871; Requirelogon=1;"
```

- Using the connection attributes file

```
"Provider=Unisys.DMSII;Data Source=DBA1;"
```

- Specifying a .UDL file

```
"File Name=C:\Projects\DBA\MYDB.udl;"
```

Paradigm MCP4024 34

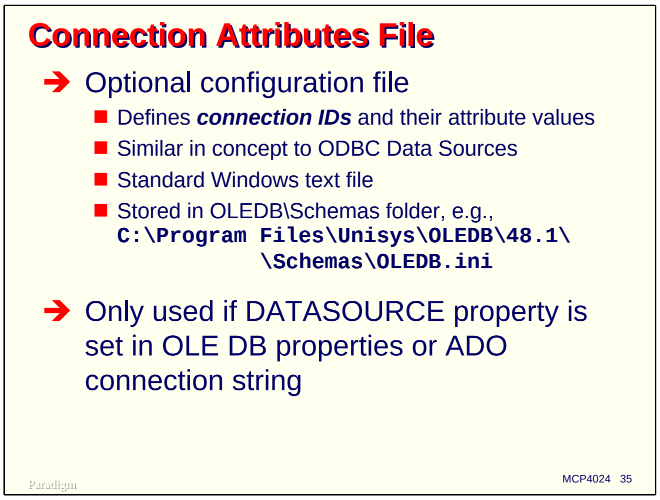
Regardless of the combination of sources from which you will supply connection attributes, you must specify at least a minimal connection string with the Open method. This slide shows the three most common ways of using a connection string.

In all cases except the UDL file, the first name/value pair in the connection string must be the identifier for the provider. For the Unisys DMSII Provider, this must be either **`Unisys.DMSII`** or **`Unisys.DMSII.1`** (they are equivalent).

In the first case, we simply supply all of the required connection attributes and a few optional ones in the connection string itself. Most of these are obvious, but I will discuss them more fully in an upcoming slide.

In the second case, the connection string specifies the name of a data source, which is an entry in the Unisys Provider's Connection Attributes file. This name is termed the ***Connection ID***. Associated with this Connection ID are a set of connection attributes. There can be any number of Connection IDs defined.

The third case shows how to reference a UDL file. The connection attributes will simply be read from this file.



Connection Attributes File

- Optional configuration file
  - Defines ***connection IDs*** and their attribute values
  - Similar in concept to ODBC Data Sources
  - Standard Windows text file
  - Stored in OLEDB\Schemas folder, e.g., **`C:\Program Files\Unisys\OLEDB\48.1\`** **`\Schemas\OLEDB.ini`**
- Only used if DATASOURCE property is set in OLE DB properties or ADO connection string

Paradigm MCP4024 35

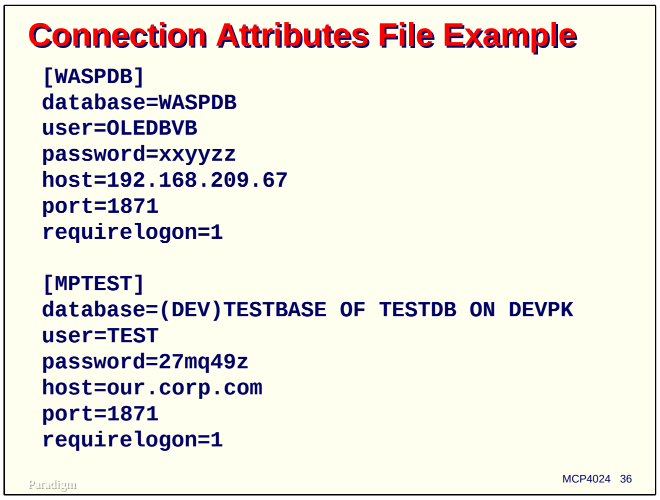
The Connection Attributes file is used very often with the Unisys Provider, especially when the data consumer is running on an unattended server. It serves much the same function as do ODBC Data Sources in the ODBC Administrator. The file is a simple Windows text file—you can edit it with Notepad or any other Windows editor. The file is stored in the Provider's **`Schemas`** folder along with the rest of the software installed for the provider. By default, this is in the Unisys folder under Program Files.

The Connection Attributes file is referenced at open time only if the DATASOURCE property is set in the OLE DB or ADO object, or you use the Data Source attribute in a connection string.

The file consists of a set of entries, prefixed by a Connection ID in square brackets, and followed by the attributes in name/value pair form, one per line. See the next slide for an example. As of MCP 8.0, you can specify the following attributes:

- **`database`** = <MCP data base title>
- **`user`** = <MCP user ID fields>
- **`password`** = <MCP password>
- **`host`** = <computer ID>
- **`port`** = <port number>
- **`requirelogon`** = <Boolean>
- **`aliasccs`** = <MCP Coded Character Set name>
- **`controlfile`** = <data base control file title>
- **`hidechapters`** = <Boolean>
- **`schemalocation`** <local schema file path>
- **`physicaldbonly`** = <Boolean>
- **`concealindexes`** = <Boolean>
- **`normalize`** = <Boolean>

See the Provider *Operations Guide* for complete information on all of these attributes.



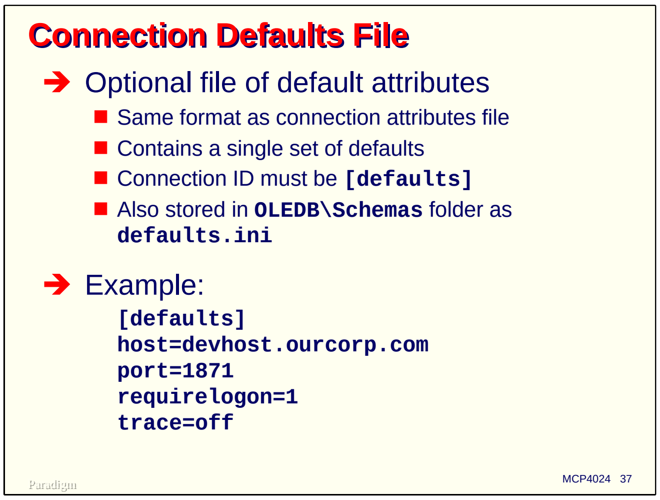
Connection Attributes File Example

```
[WASPDB]
database=WASPDB
user=OLEDBVB
password=xxyyzz
host=192.168.209.67
port=1871
requirelogon=1

[MPTEST]
database=(DEV)TESTBASE OF TESTDB ON DEVPK
user=TEST
password=27mq49z
host=our.corp.com
port=1871
requirelogon=1
```

Paradigm MCP4024 36

This slide shows a simple Connection Attributes file with two sets of Connection IDs and their attributes.

Note the second example, which illustrates how to reference a logical database within a DMSII description. **`TESTBASE`** is the name of the logical database; **`TESTDB`** is the name of the data base.



Connection Defaults File

- Optional file of default attributes
  - Same format as connection attributes file
  - Contains a single set of defaults
  - Connection ID must be **`[defaults]`**
  - Also stored in **`OLEDB\Schemas`** folder as **`defaults.ini`**
- Example:

```
[defaults]
host=devhost.ourcorp.com
port=1871
requirelogon=1
trace=off
```

Paradigm MCP4024 37

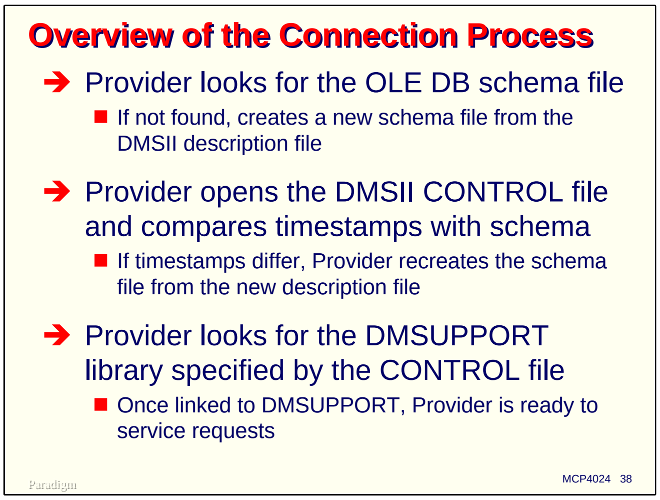
The Connection Defaults file is another file of connection attributes stored in the Schemas folder of the Windows environment. It contains a single set of default attribute values. If attributes are not specified in the Connection Attributes file (or at a higher level), they will be picked up from the defaults file.

The defaults file is another standard Windows text file with the same structure as the Connection Attributes file. It must have a single Connection ID, **`[defaults]`**.

The defaults file can contain all of the attributes of the Connection Attributes file, plus some additional ones that apply to operation of the Unisys Provider as a whole:

- **`trace`** = on | off | bit mask
- **`securetrace`** = on | off
- **`timeout`** = <integer> [in seconds]
- **`nodisplay`** = 0 | 1
- **`readonly`** = <Boolean>
- **`cloak`** = on | off | <integer>
- **`transactiontimeout`** = <integer>
- **`unisysinitproperties`** = <Boolean>

See the Provider *Operations Guide* for more information on these special attributes for the defaults file.



Overview of the Connection Process

- ➔ Provider looks for the OLE DB schema file
  - ■ If not found, creates a new schema file from the DMSII description file
- ➔ Provider opens the DMSII CONTROL file and compares timestamps with schema
  - ■ If timestamps differ, Provider recreates the schema file from the new description file
- ➔ Provider looks for the DMSUPPORT library specified by the CONTROL file
  - ■ Once linked to DMSUPPORT, Provider is ready to service requests

Paradigm MCP4024 38

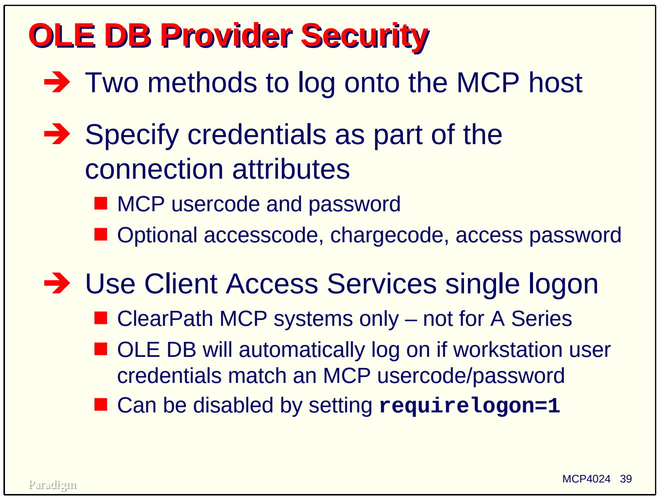
The Unisys Provider for DMSII goes through a number of steps during the process of opening a connection to locate the schema file, validate it, generate it if necessary, and locate the DMSII files for the data base. All of these steps take place in MCP environment.

First, the Provider looks for the master copy of the OLE DB schema file. If not found, it generates a new schema file from the data base description file and stores it under the same usercode and family. This is why the **`database`** attribute in a Connection Attributes must identify the location and name of the description file.

Next, the Provider attempts to open the data base control file, the location of which it got from the description file (and which is stored in the schema file). You can override this location in the Connection Attributes file, if you need to. The Provider compares the data base timestamp in the control file with the one in the schema file. If they are the same, the process continues. If they are different, it means the data base schema has been changed (i.e., DASDL update) since the last time a connection was established. In this case the master schema file is regenerated from the updated description file.

Once the schema file is in sync with the control file, the Provider looks for the DMSUPPORT library (whose location is stored in the control file) and links to it. After the linkage is established, the Provider is ready to service OLE DB requests.

The process works very well if you use the standard DMSII naming conventions and file location conventions. If you description file does not accurately specify the location of the control file or DMSUPPORT library, however, you can suffer NO FILE conditions during the open process. The Provider hangs during these NO FILEs, so it is a good idea to avoid them.



OLE DB Provider Security

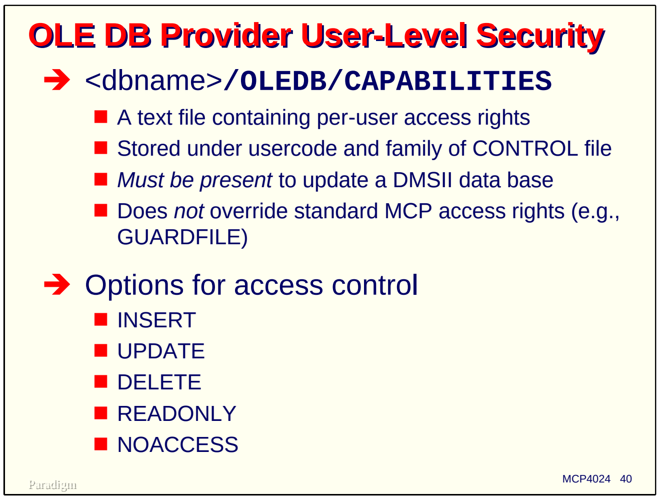
- Two methods to log onto the MCP host
- Specify credentials as part of the connection attributes
  - MCP usercode and password
  - Optional accesscode, chargecode, access password
- Use Client Access Services single logon
  - ClearPath MCP systems only – not for A Series
  - OLE DB will automatically log on if workstation user credentials match an MCP usercode/password
  - Can be disabled by setting **`requirelogon=1`**

Paradigm

MCP4024 39

The Unisys Provider for DMSII fully supports and adheres to MCP file security on the data base and related software files. There are two methods by which the client system can authenticate itself.

- You can supply an MCP usercode and password in the connection string (or in the Connection Attributes file). You can also supply a chargecode, accesscode, and access password if your site uses them.
- If the MCP host is a ClearPath system, you can use the single logon feature of Client Access Services (formerly NX/Services). The Windows system will pass through the user credentials and the Provider will use them to authenticate the connection to the data base. If you want to use different credentials than the ones used by Client Access Services, you can set the **`requirelogon`** attribute to **`1`** for the connection.



OLE DB Provider User-Level Security

- <dbname>**/OLEDB/CAPABILITIES**
  - A text file containing per-user access rights
  - Stored under usercode and family of CONTROL file
  - *Must be present* to update a DMSII data base
  - Does *not* override standard MCP access rights (e.g., GUARDFILE)
- Options for access control
  - INSERT
  - UPDATE
  - DELETE
  - READONLY
  - NOACCESS

Paradigm

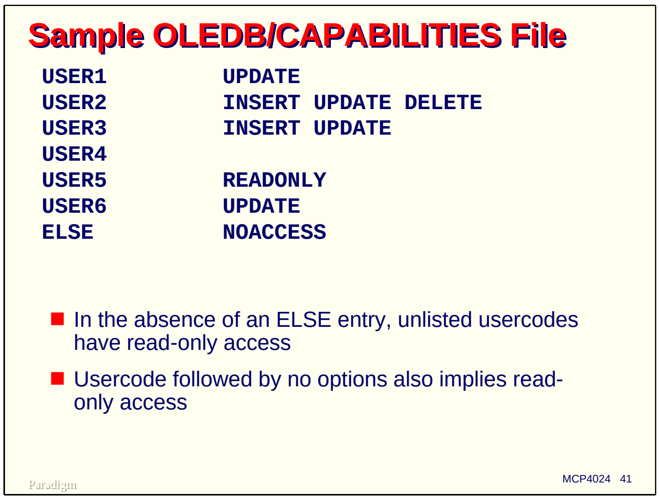
MCP4024 40

In addition to fully supporting MCP file security for the data base, the OLE DB Provider has an internal mechanism to restrict access on a user-by-user basis. These restrictions are in addition to those for standard MCP file security, not in place of them.

These user-level access rights are stored in a file titled <data base name>**/OLEDB/CAPABILITIES** stored under the same usercode and family as the data base control file. It is a standard MCP text file and can be edited using CANDE, EDITOR, or PWB.

It is important to note that *this file must be present* in order for OLE DB users to update the data base. If this file is not present, then all OLE DB users will have, at most, read-only access to the data base.

There are five options which can be associated with each user in this file. **INSERT**, **UPDATE**, and **DELETE** can be specified in any combination and indicate which update rights the user will have. If **READONLY** is specified, the user will have read-only access through OLE DB. If **NOACCESS** is specified, the user will not be allowed to access the data base through OLE DB.



Sample OLEDB/CAPABILITIES File

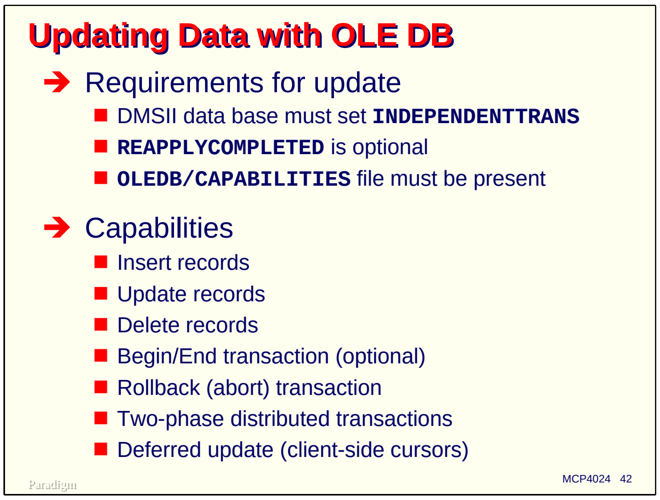
```
USER1          UPDATE
USER2          INSERT UPDATE DELETE
USER3          INSERT UPDATE
USER4
USER5          READONLY
USER6          UPDATE
ELSE           NOACCESS
```

- In the absence of an ELSE entry, unlisted usercodes have read-only access
- Usercode followed by no options also implies read-only access

Paradigm MCP4024 41

This slide shows a sample of what an **OLEDB/CAPABILITIES** file can contain. Each user is specified on a separate record. The entries are free form, with the usercode as the first token on a line, followed by a list of that user's access rights.

A special entry, **ELSE**, specifies the access rights for all users not otherwise mentioned in the file. If a usercode is specified without any options, it will have read-only access through OLE DB.



Updating Data with OLE DB

- Requirements for update
  - DMSII data base must set **INDEPENDENTTRANS**
  - **REAPPLYCOMPLETED** is optional
  - **OLEDB/CAPABILITIES** file must be present
- Capabilities
  - Insert records
  - Update records
  - Delete records
  - Begin/End transaction (optional)
  - Rollback (abort) transaction
  - Two-phase distributed transactions
  - Deferred update (client-side cursors)

Paradigm MCP4024 42

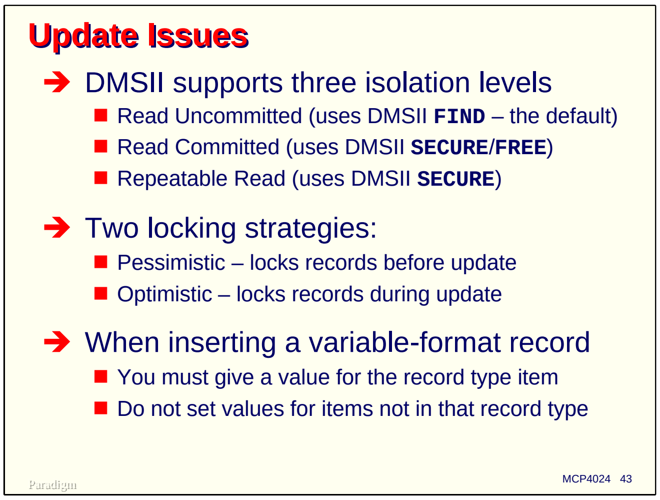
The OLE DB Provider for DMSII supports a large subset of the possible update operations. In order to enable updates through OLE DB, some requirements must be met first:

- The data base must have the **INDEPENDENTTRANS** option set in the DASDL. Data bases without **INDEPENDENTTRANS** will be treated as read only by the Provider. The **REAPPLYCOMPLETED** option is optional unless you are doing distributed updates, but recommended in any case.
- An **OLEDB/CAPABILITIES** file must be present under the usercode and family of the data base control file. In the absence of this file, all OLE DB users will have read-only access.

Through the Provider you can add, delete, and update records in the data base. You can optionally declare begin- and end-transactions to enclose multiple update operations as a monolithic transaction. End-transaction in OLE DB and ADO is referred to as a "commit". You can roll back (abort) a transaction in mid-stream.

It is also possible for the Provider to participate in two-phase distributed transactions, such as those available through SQL Server. In order to do this, you must set the **OPENOLTP** and **REAPPLYCOMPLETED** options in DASDL, and generate the RMSUPPORT library for the data base.

Finally, the Provider now supports deferred update through ADO client-side cursors and the UpdateBatch method.



Update Issues

- DMSII supports three isolation levels
  - Read Uncommitted (uses DMSII **FIND** – the default)
  - Read Committed (uses DMSII **SECURE**/**FREE**)
  - Repeatable Read (uses DMSII **SECURE**)
- Two locking strategies:
  - Pessimistic – locks records before update
  - Optimistic – locks records during update
- When inserting a variable-format record
  - You must give a value for the record type item
  - Do not set values for items not in that record type

Paradigm MCP4024 43

Other data base systems, especially relational ones, use special terms to describe how record locking will occur during queries and update transactions. OLE DB has adopted these terms, but there are translations of them to equivalent DMSII terms.

Isolation level describes how protected a query or update transaction will be from other updates occurring simultaneously against the data base. This is a property which is set for the connection as a whole. DMSII supports the first three levels:

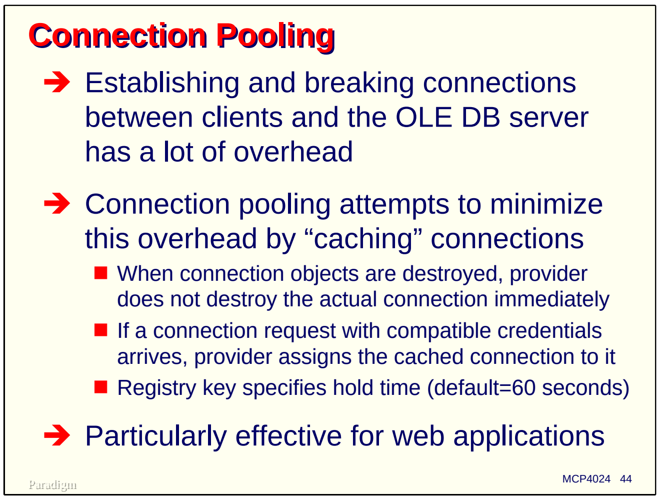
- **Read Uncommitted** is similar to an ordinary DMSII FIND. The record is read regardless of who may be updating it at that moment.
- **Read Committed** assures that no other process is updating the record at the time it is read. This is equivalent to a DMSII SECURE followed by a FREE.
- **Repeatable Read** assures that no other process is updating the record throughout the transaction in which the read occurred. This is equivalent to a DMSII SECURE (which will be freed automatically at end-transaction).

The Provider does not yet support the fourth level of isolation, **Serializable**. That capability has been announced for MCP 9.0.

The locking strategy indicates when records should be locked during a transaction. This is a property which is set on individual recordsets.

- Pessimistic locking assumes that the records may not be available during the transaction, so attempts to lock them before posting any updates to the data.
- Optimistic locking assumes that the records will be available, so locks them as necessary during the transaction. If during the transaction a record turns out not to be available, the transaction aborts, which requires rolling back all updates done up to that point. Since in most cases transactions succeed, this method usually provides much better performance.

There is one final update-related issue. When you insert a record into a variable-format data set, you must specify the value of the record type item at the time the record is added. OLE DB and ADO do not have the equivalent to a DMSII CREATE command, so specifying the record type item in the initial insert is the only way that DMSII can determine which record format to allocate in the data set.



Connection Pooling

- ➔ Establishing and breaking connections between clients and the OLE DB server has a lot of overhead
- ➔ Connection pooling attempts to minimize this overhead by “caching” connections
  - When connection objects are destroyed, provider does not destroy the actual connection immediately
  - If a connection request with compatible credentials arrives, provider assigns the cached connection to it
  - Registry key specifies hold time (default=60 seconds)
- ➔ Particularly effective for web applications

Paradigm MCP4024 44

The OLE DB Provider supports connection pooling. This is a standard capability Microsoft has built into OLE DB. Its purpose is to reduce the considerable overhead involved in establishing and terminating connections between clients and data sources.

Connection pooling works by caching connections. When a client terminates its OLE DB connection, the connection is left open and put in a pool of other open but inactive connections. These connections are held in the pool for a specified period of time, 60 seconds by default.

If a compatible connection request arrives while one of these cached connections is in the pool, it is retrieved from the pool and associated with the new request. The overhead for caching and re-associating connections is many times less than that for creating a new connection and completely destroying it at termination time.

By "compatible connection" I mean one that matches the attributes of the original connection. The host name, data base name, user credentials, and other attributes in the new request must be the same as those in the request which originally created the connection.

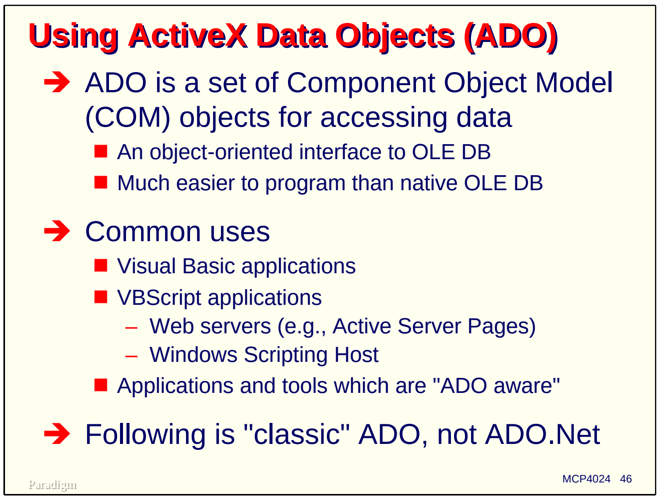
The hold time for cached connections is specified in a Windows registry key. See the Provider *Operations Guide* for this key's GUID.

Connection pooling is especially effective in environments where connections are established and terminated very quickly. Prime examples of this are web applications, such as those based on Active Server Pages with ADO, which generally make and break connections as each HTTP request arrives.

Applications of OLE DB

Paradigm

With that background on OLE DB and establishing connections to data bases, let us now look at some ways in which it can be used.



Using ActiveX Data Objects (ADO)

- ADO is a set of Component Object Model (COM) objects for accessing data
  - An object-oriented interface to OLE DB
  - Much easier to program than native OLE DB
- Common uses
  - Visual Basic applications
  - VBScript applications
    - Web servers (e.g., Active Server Pages)
    - Windows Scripting Host
  - Applications and tools which are "ADO aware"
- Following is "classic" ADO, not ADO.Net

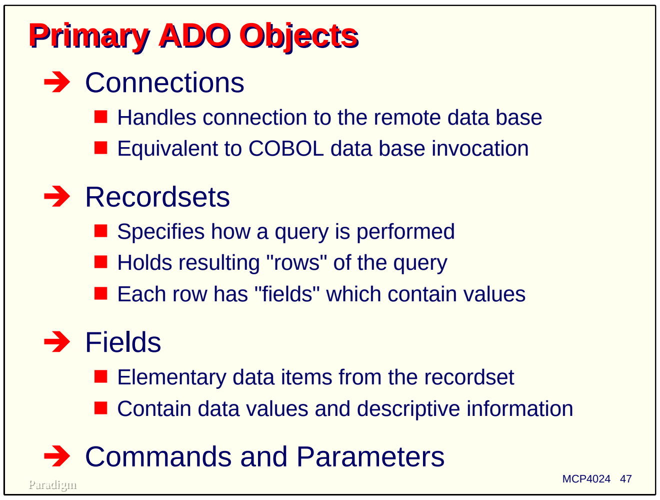
Paradigm MCP4024 46

One of the primary ways you can use OLE DB is through ActiveX Data Objects, or ADO. As mentioned earlier, ADO is an object-oriented front-end to OLE DB. It is much easier to program for than the native OLE DB interfaces, and works with more approachable programming environments than C++, such as Visual Basic.

If you are using one of the flavors of Visual Basic, ADO is almost certainly the mechanism you will use to access data on remote hosts. It works especially well with VBScript, which is the language most often used with Active Server Pages and the Windows Scripting Host. There are also a number of applications and tools which are "ADO aware" and can use it to connect to OLE DB providers.

Do not confuse ADO with DAO. DAO (Data Access Objects) is also an object-oriented interface for data bases, but it interfaces with the Jet data base engine used with Microsoft Access (i.e., **`.mdb`** files). Microsoft has an OLE DB provider for Jet, so you can use ADO with Jet data bases and Access applications instead of DAO.

Also, do not confuse the "classic" ADO being presented here with ADO.Net. ADO.Net is based somewhat on classic ADO, but they are entirely different things.



Primary ADO Objects

- Connections
  - Handles connection to the remote data base
  - Equivalent to COBOL data base invocation
- Recordsets
  - Specifies how a query is performed
  - Holds resulting "rows" of the query
  - Each row has "fields" which contain values
- Fields
  - Elementary data items from the recordset
  - Contain data values and descriptive information
- Commands and Parameters

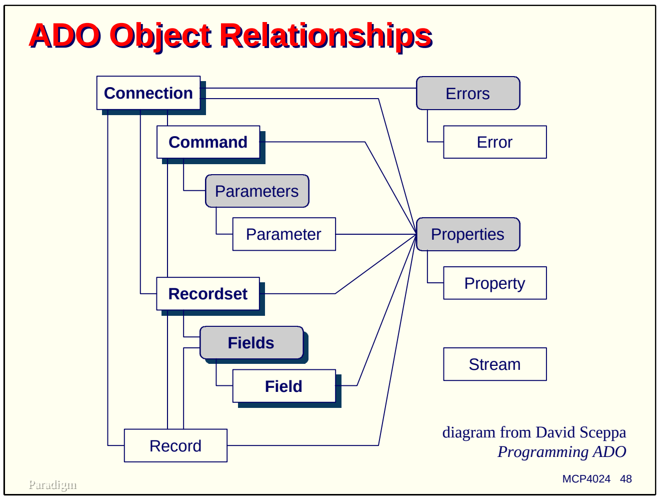
Paradigm

MCP4024 47

The ADO programming model uses a number of types of objects. There are three that you use almost all of the time.

- Connections establish connections between clients and OLE DB data sources. They are somewhat equivalent to a COBOL data base invocation.
- Recordsets provide a means to specify how a query or update operation will take place and a mechanism for returning the results of a query. ADO recordsets map directly to OLE DB rowsets. A recordset typically contains an arbitrary number of rows (records). Each row consists of a number of fields.
- Fields are the elementary data items for a query and hold the actual data values.

There are two other ADO objects that are frequently used, especially when connecting to data bases that support stored procedures. Commands can be used to submit query criteria, which may be an SQL statement or a reference to a stored procedure. Parameters are objects which supply values to the arguments of a query or stored procedure.



ADO Object Relationships

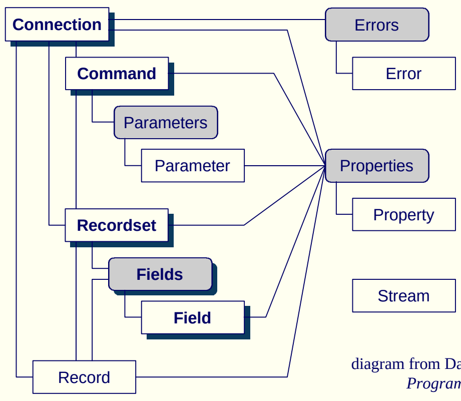
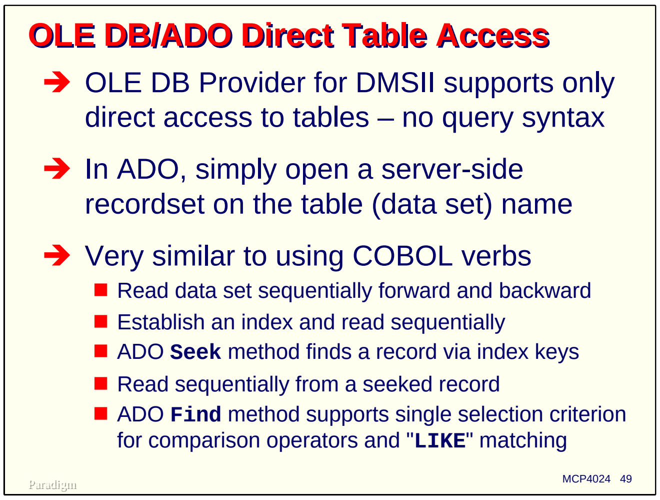


diagram from David Sceppa *Programming ADO*

Paradigm MCP4024 48

This diagram comes from David Sceppa's book, *Programming ADO*. It clearly shows the relationships among ADO objects. The square rectangles represent objects in the ADO model. The rounded rectangles represent collections, which hold arbitrary numbers of a certain type of object.



OLE DB/ADO Direct Table Access

- OLE DB Provider for DMSII supports only direct access to tables – no query syntax
- In ADO, simply open a server-side recordset on the table (data set) name
- Very similar to using COBOL verbs
  - Read data set sequentially forward and backward
  - Establish an index and read sequentially
  - ADO **`Seek`** method finds a record via index keys
  - Read sequentially from a seeked record
  - ADO **`Find`** method supports single selection criterion for comparison operators and "**`LIKE`**" matching

Paradigm MCP4024 49

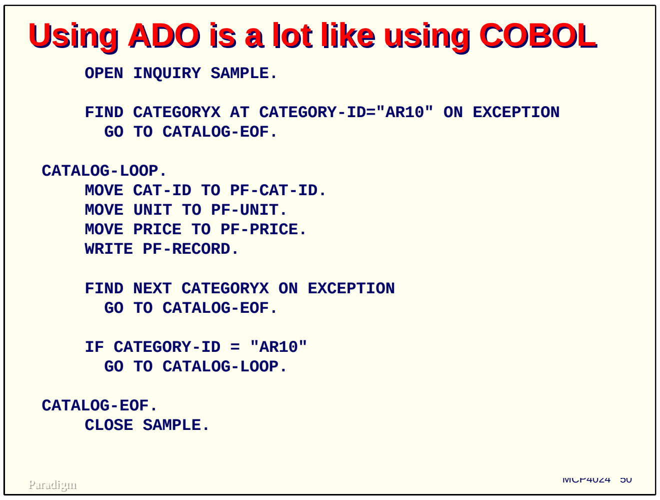
When you use OLE DB or ADO to connect directly to a DMSII data source, you use a mode of query known as Direct Table Access. Since the DMSII Provider does not support the OLE DB "command" interface, you cannot pose queries to DMSII the same way you can to, say, SQL Server.

In ADO, Direct Table Access means you open the entire table. It does not get copied to your client system (unless you use a client-side cursor). It's more like opening a file, where you get access to the file but have to look at it a record (row) at a time.

With Direct Table Access, you can read the data set sequentially. You can specify an index and read the data set sequentially in the key order of that index. You can also use an index to seek to a record with specified key values. You can then read sequentially from that record via the index.

Using ADO in this manner is very much like programming for DMSII in COBOL. There are a couple of nice additional things you can do in ADO, such as use the Find method to filter records on a single selection criterion and use substring matching similar to the SQL **`LIKE`** predicate.

We will look at some examples of simple ADO usage next.



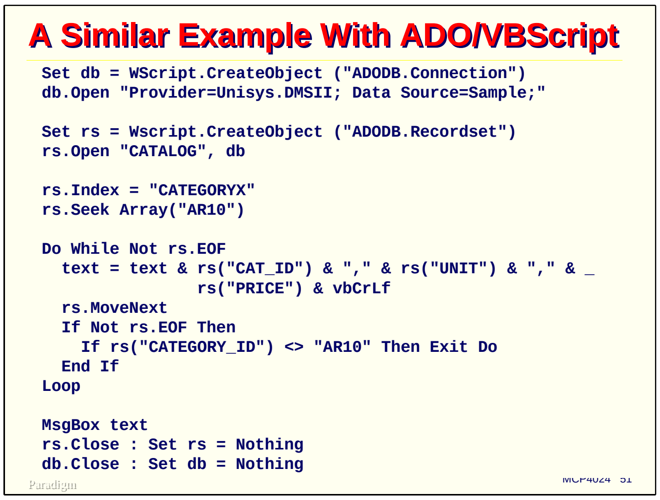
Using ADO is a lot like using COBOL

```
     OPEN INQUIRY SAMPLE.

     FIND CATEGORYX AT CATEGORY-ID="AR10" ON EXCEPTION
       GO TO CATALOG-EOF.

CATALOG-LOOP.
     MOVE CAT-ID TO PF-CAT-ID.
     MOVE UNIT TO PF-UNIT.
     MOVE PRICE TO PF-PRICE.
     WRITE PF-RECORD.

     FIND NEXT CATEGORYX ON EXCEPTION
       GO TO CATALOG-EOF.

     IF CATEGORY-ID = "AR10"
       GO TO CATALOG-LOOP.

CATALOG-EOF.
     CLOSE SAMPLE.
```

Paradigm MCP4024 50

This slide shows a snippet of COBOL doing a very common DMSII task—finding a record via an index and then reading via that index sequentially until some terminating condition is encountered.

The snippet opens the data base and does a FIND to locate the first record of interest. It then goes into a loop, which formats some fields from the current record, advances to the next record, and tests for the terminating condition. When that condition occurs, the loop exits and the data base is closed.



A Similar Example With ADO/VBScript

```
Set db = WScript.CreateObject ("ADODB.Connection")
db.Open "Provider=Unisys.DMSII; Data Source=Sample;"

Set rs = Wscript.CreateObject ("ADODB.Recordset")
rs.Open "CATALOG", db

rs.Index = "CATEGORYX"
rs.Seek Array("AR10")

Do While Not rs.EOF
  text = text & rs("CAT_ID") & "," & rs("UNIT") & "," & _
              rs("PRICE") & vbCrLf
  rs.MoveNext
  If Not rs.EOF Then
    If rs("CATEGORY_ID") <> "AR10" Then Exit Do
  End If
Loop

MsgBox text
rs.Close : Set rs = Nothing
db.Close : Set db = Nothing
```

Paradigm MCP4024 51

This slide shows a snippet of VBScript that does essentially the same thing.

First we create an ADO Connection object and open it using a connection string which references a Connection ID in the DMSII Provider's Connection Attributes file.

Next, we create a Recordset object and open it, specifying the **`CATALOG`** table and associating the Recordset with the Connection object.

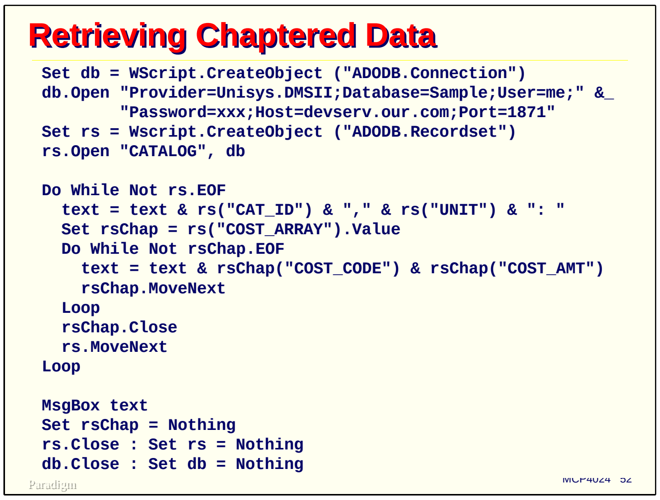
After opening the Recordset, we establish which index for the data set we want to use and seek to a record with a specified key. The Seek method takes an array parameter which contains the key values for the index. In this case the index has only one key item, so we could have omitted the Array constructor and just used the value by itself.

Now the script goes into a loop, formatting some fields from the current row of the Recordset. If we are at the end of the Recordset (or if the Recordset had no rows) the EOF property will be true, and the loop will terminate. Fields are fetched from Recordset rows by specifying their name as a string in parentheses after the Recordset identifier. Note that hyphens in the original DASDL names are replaced by underscores when names are referenced through OLE DB.

The MoveNext method advances the Recordset to the next row, and if necessary, sets the EOF property. The script tests for EOF and the terminating key condition. In either case the loop exits.

Finally, the script closes the Recordset and Connection objects, and deallocates them by setting their object references to the VB keyword **`Nothing`**.

Cognoscente of ADO will be quick to point out that this script is not very well written and will execute quite inefficiently. That is true, and it was presented this way more to illustrate ADO capabilities than good coding practice. Besides, nothing else would fit on this slide.



Retrieving Chaptered Data

```
Set db = WScript.CreateObject ("ADODB.Connection")
db.Open "Provider=Unisys.DMSII;Database=Sample;User=me;" &_
        "Password=xxx;Host=devserv.our.com;Port=1871"
Set rs = Wscript.CreateObject ("ADODB.Recordset")
rs.Open "CATALOG", db

Do While Not rs.EOF
  text = text & rs("CAT_ID") & "," & rs("UNIT") & ": "
  Set rsChap = rs("COST_ARRAY").Value
  Do While Not rsChap.EOF
    text = text & rsChap("COST_CODE") & rsChap("COST_AMT")
    rsChap.MoveNext
  Loop
  rsChap.Close
  rs.MoveNext
Loop

MsgBox text
Set rsChap = Nothing
rs.Close : Set rs = Nothing
db.Close : Set db = Nothing
```

Paradigm MCP4024 52

I have previously discussed how occurring items and embedded data sets can be represented in OLE DB as chaptered rowsets. This script illustrates how you can access these chapters using ADO.

We start out the same way as with the prior example, creating and opening both Connection and Recordset objects. The only difference is that this time we have specified the connection attributes in the connection string, rather than referencing a Connection ID in the Provider's Connection Attributes file.

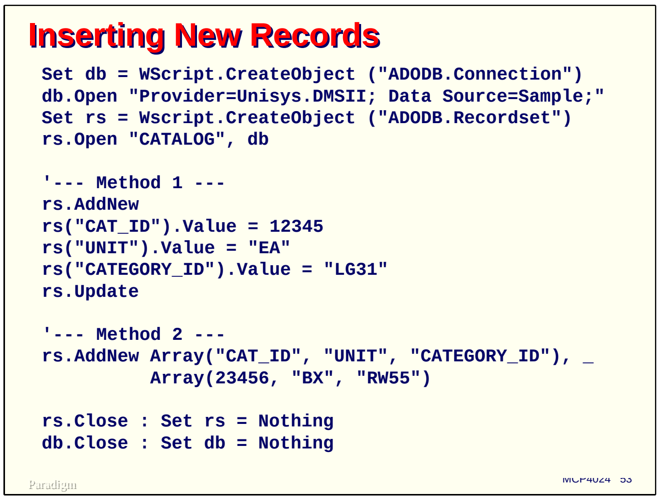
Assume that our **`CATALOG`** data set contains an occurring group named **`COST-ARRAY`**, which in turn has at least two subordinate items, **`COST-CODE`** and **`COST-AMT`**.

The script enters an outer loop for the Recordset as before and formats a couple of the ordinary data items from it. The field for the occurring group is not an ordinary item, however. It is instead a reference to a chapter Recordset object that holds the data for the occurring item. Each occurrence corresponds to one row of this Recordset object, and the fields of each row correspond to the elementary data items of the occurring group.

The script therefore assigns an object reference to the chapter item. It then enters an inner loop to fetch the rows of the chapter and the values of its fields from each row. Note that you cannot address the entire array at once, you can only see one row of it at a time, just as with records in a file.

At the end of the inner loop, the script closes the chapter Recordset and moves to the next row of the main Recordset for the data set. At the end of the script, the chapter object is deallocated along with the others.

You cannot actually tell from this script that the chapter represents an array. It could as easily have been an embedded data set. The chapter coding in ADO for both occurring items and embedded structures is the same.



Inserting New Records

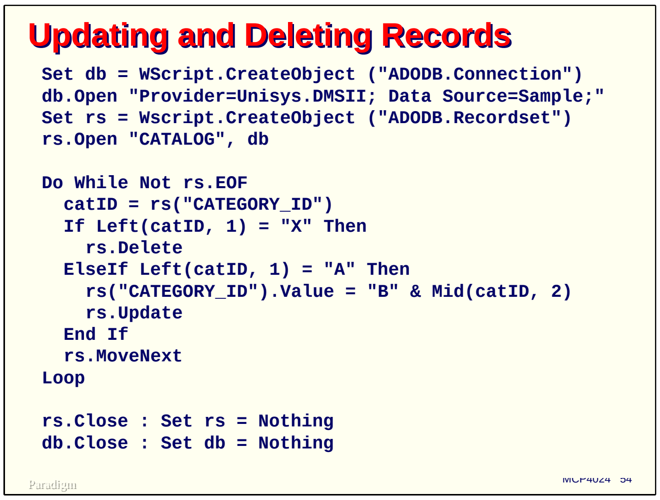
```
Set db = WScript.CreateObject ("ADODB.Connection")
db.Open "Provider=Unisys.DMSII; Data Source=Sample;"
Set rs = Wscript.CreateObject ("ADODB.Recordset")
rs.Open "CATALOG", db

'--- Method 1 ---
rs.AddNew
rs("CAT_ID").Value = 12345
rs("UNIT").Value = "EA"
rs("CATEGORY_ID").Value = "LG31"
rs.Update

'--- Method 2 ---
rs.AddNew Array("CAT_ID", "UNIT", "CATEGORY_ID"), _
          Array(23456, "BX", "RW55")

rs.Close : Set rs = Nothing
db.Close : Set db = Nothing
```

Paradigm MCP4024 53

ADO offers two ways to insert records into a data set.

In the first way, you call the AddNew method of the Recordset to signal that you are creating a new row. Then you store values into fields of the Recordset. When all of the fields have been set, you call the Update method of the Recordset to store the record. This is very similar to the CREATE/STORE sequence you would use in a COBOL program.

In the second way, you call the AddNew method, passing two array parameters. The first array contains a list of item names; the second array contains a corresponding list of values to be stored in those items. You *must* use this technique to insert records in a variable-format data set, and the record type item for the data set *must* be assigned a value in the AddNew call.



Updating and Deleting Records

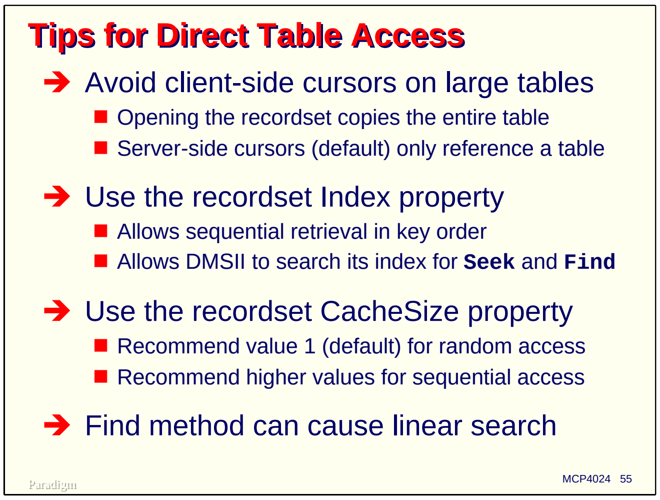
```
Set db = WScript.CreateObject ("ADODB.Connection")
db.Open "Provider=Unisys.DMSII; Data Source=Sample;"
Set rs = Wscript.CreateObject ("ADODB.Recordset")
rs.Open "CATALOG", db

Do While Not rs.EOF
  catID = rs("CATEGORY_ID")
  If Left(catID, 1) = "X" Then
    rs.Delete
  ElseIf Left(catID, 1) = "A" Then
    rs("CATEGORY_ID").Value = "B" & Mid(catID, 2)
    rs.Update
  End If
  rs.MoveNext
Loop

rs.Close : Set rs = Nothing
db.Close : Set db = Nothing
```

Paradigm MCP4024 54

This script is a very contrived example, but it shows how to update and delete records using ADO.

The script loops through all of the records in the **`CATALOG`** data set, examining the first character of the **`CATEGORY-ID`** item. If this character is an "**`X`**", it deletes the record. If this character is an "**`A`**", it replaces the "**`A`**" with a "**`B`**" and updates the record with this new field value.

DAO programmers should note that there is no Edit method in ADO. To update a record you simply make it the current row, set new field values, and call the Update method.



Tips for Direct Table Access

- Avoid client-side cursors on large tables
  - Opening the recordset copies the entire table
  - Server-side cursors (default) only reference a table
- Use the recordset Index property
  - Allows sequential retrieval in key order
  - Allows DMSII to search its index for **`Seek`** and **`Find`**
- Use the recordset CacheSize property
  - Recommend value 1 (default) for random access
  - Recommend higher values for sequential access
- Find method can cause linear search

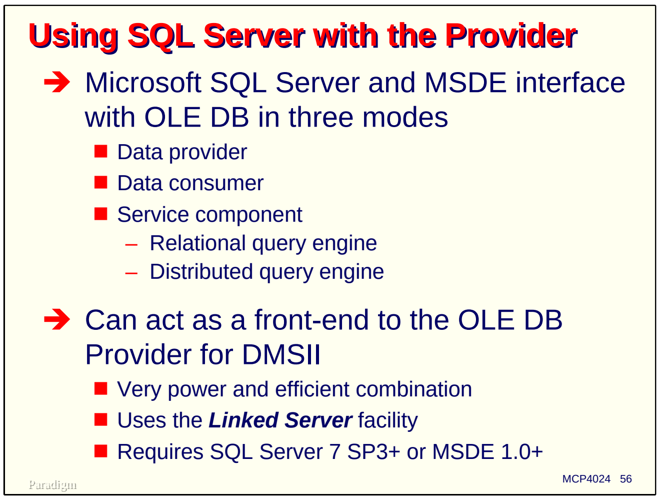
Paradigm MCP4024 55

Here are several tips to keep in mind when using Direct Table Access.

- Avoid using client-side cursors, especially on large data sets. When you open a client-side cursor, OLE DB obligingly copies the entire data set to the client. This could take a while, and it could blow away your client by consuming all of the resources on the system. Server-side cursors are the default, and only open a reference to the table.
- Use the Recordset Index property. This takes advantage of the integrated index feature of OLE DB. It allows you to retrieve records sequentially in order by the index key items, and with the Seek and Find methods, randomly access records in the data set.
- If you will be retrieving a large number of records sequentially, set the CacheSize property to a number greater than one. This enables blocking and caching of rows in the transport layer, which can significantly improve performance. For random retrievals, leave the CacheSize at one, its default value. If you are using MDAC levels below 2.6, you need to use special coding when setting the CacheSize higher than 1 and using the Seek method. The Unisys documentation recommends one of these two approaches:

  -
    ```
    cachesize = 1
    open table
    set index
    cachesize = n
    seek
    ```

  -
    ```
    set index
    open table
    cachesize = 1
    movefirst
    cachesize = n
    seek
    ```
- Be aware that certain uses of the Find method can cause DMSII to enter a linear search. These are generally time consuming and cannot be DS-ed.



Using SQL Server with the Provider

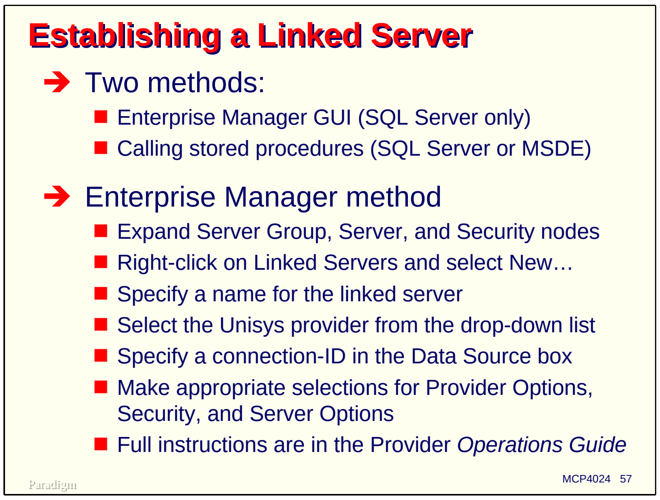
- Microsoft SQL Server and MSDE interface with OLE DB in three modes
  - Data provider
  - Data consumer
  - Service component
    - Relational query engine
    - Distributed query engine
- Can act as a front-end to the OLE DB Provider for DMSII
  - Very power and efficient combination
  - Uses the ***Linked Server*** facility
  - Requires SQL Server 7 SP3+ or MSDE 1.0+

Paradigm MCP4024 56

Microsoft SQL Server (and its somewhat restricted desktop version, MSDE) can interface with OLE DB providers in three ways—as a data provider, supplying data from its own data bases, as a data consumer, retrieving data form other OLE DB data sources, and as a service component. SQL Server can supply two services to OLE DB consumers and providers: acting as a relational query engine and acting as a distributed query engine.

SQL Server works especially well with the Unisys OLE DB Provider for DMSII. SQL Server can process queries in its dialect of SQL, decompose them into more primitive OLE DB operations, and pass those operations to the Unisys Provider for execution. Effectively, SQL Server becomes a query processor for DMSII. This is a very powerful and efficient combination.

SQL Server interfaces to DMSII through SQL's Linked Server facility. A linked server is an OLE DB data source that SQL Server can use much as if it were one of SQL's own data bases. In order to configure linked servers, you must be using at least SQL Server 7 SP3, or MSDE version 1.0. Linked servers work somewhat better in SQL Server 2000, but for use with DMSII you should be on 2000 SP1 or later. SP1 fixed some serious bugs in SQL 2000 that prevented many DMSII queries from functioning properly.



Establishing a Linked Server

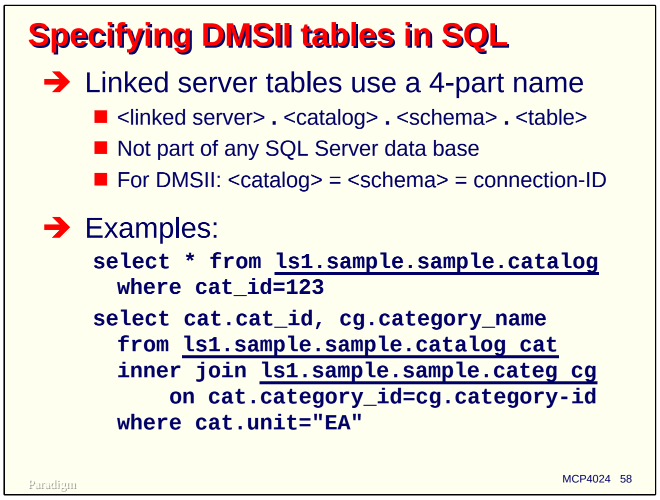
- Two methods:
  - Enterprise Manager GUI (SQL Server only)
  - Calling stored procedures (SQL Server or MSDE)
- Enterprise Manager method
  - Expand Server Group, Server, and Security nodes
  - Right-click on Linked Servers and select New…
  - Specify a name for the linked server
  - Select the Unisys provider from the drop-down list
  - Specify a connection-ID in the Data Source box
  - Make appropriate selections for Provider Options, Security, and Server Options
  - Full instructions are in the Provider *Operations Guide*

Paradigm MCP4024 57

You can establish a linked server in SQL Server in one of two ways.

- The easiest is to use the Enterprise Manager GUI. It provides a series of dialogs that allow you to enter the necessary configuration information. The slide shows a brief outline of the steps you need to take. The process generally takes less than two minutes.
- You can also use stored procedures. This is the required method with MSDE, as it does not include the Enterprise Manager.

Full instructions for configuring the OLE DB Provider for DMSII as a linked server are contained in the Provider's *Operations Guide*.



Specifying DMSII tables in SQL

- Linked server tables use a 4-part name
  - <linked server> . <catalog> . <schema> . <table>
  - Not part of any SQL Server data base
  - For DMSII: <catalog> = <schema> = connection-ID
- Examples:

```
select * from ls1.sample.sample.catalog
  where cat_id=123
select cat.cat_id, cg.category_name
  from ls1.sample.sample.catalog cat
  inner join ls1.sample.sample.categ cg
      on cat.category_id=cg.category-id
  where cat.unit="EA"
```

Paradigm MCP4024 58

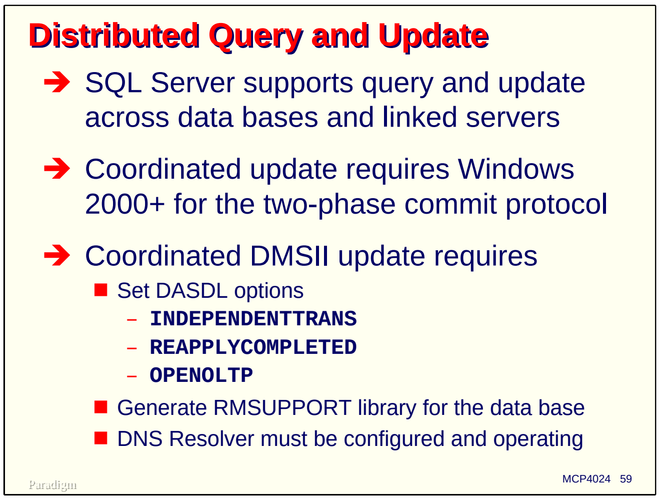
A linked server allows DMSII to act like it's part of SQL Server, but it's not, really. Because of that you need to use a special convention when referencing tables (data sets) exposed by the OLE DB Provider.

Normally tables are identified in SQL Server using a three-level name, consisting of the catalog name, the schema name, and the table name within that schema. In most cases, the catalog and schema names are set as defaults (e.g., through the USE statement) and only the table name is required.

When referencing tables for a linked server in T-SQL (the dialect of SQL used with SQL Server), you must use a four-level name. The first level is the name of the linked server as it is configured in SQL Server. The other three levels are the same as for native tables.

DMSII does not have the concept of separate catalogs and schemas. In place of these two nodes in the naming convention you must use the Connection ID of the OLE DB Provider. If you are not using the Connection Attributes file, there is no Connection ID, so you must use the data base name instead.

The example on the slide shows references to tables for a linked served named **`ls1`** which is configured to access a DMSII data base having a Connection ID of **`sample`**. In the example, the select statements reference the **`CATALOG`** and **`CATEG`** data sets of the DMSII data base. Other than the four-part name, there is nothing about these SQL queries that is different from querying native SQL Server tables. Note that you can use the standard SQL alias feature to assign shorter names to the linked server tables for other references within the query.



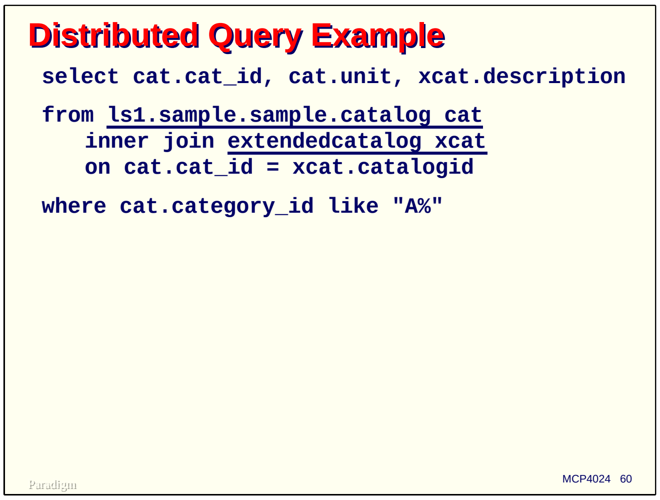
Distributed Query and Update

- SQL Server supports query and update across data bases and linked servers
- Coordinated update requires Windows 2000+ for the two-phase commit protocol
- Coordinated DMSII update requires
  - Set DASDL options
    - **`INDEPENDENTTRANS`**
    - **`REAPPLYCOMPLETED`**
    - **`OPENOLTP`**
  - Generate RMSUPPORT library for the data base
  - DNS Resolver must be configured and operating

Paradigm MCP4024 59

In addition to querying DMSII data sets through SQL Server, you can also update them. Not only that, but you can have the DMSII data base participate in a distributed update where multiple data bases (perhaps on different heterogeneous systems) are being modified within the same transaction.

Distributed update requires that you use Windows 2000 or higher, since only those versions have the necessary two-phase commit protocol implemented. You must also use the MCP 7.0 or higher version of the Provider. As mentioned earlier in the presentation, distributed update requires that the DASDL **`INDEPENDENTTRANS`**, **`REAPPLYCOMPLETED`**, and **`OPENOLTP`** options be set, and that you generate the RMSUPPORT library for the data base. Finally, the DNS Resolver must be configured and operating in the MCP environment, as the TIP protocol used by distributed update references hosts by their DNS names instead of their IP addresses.



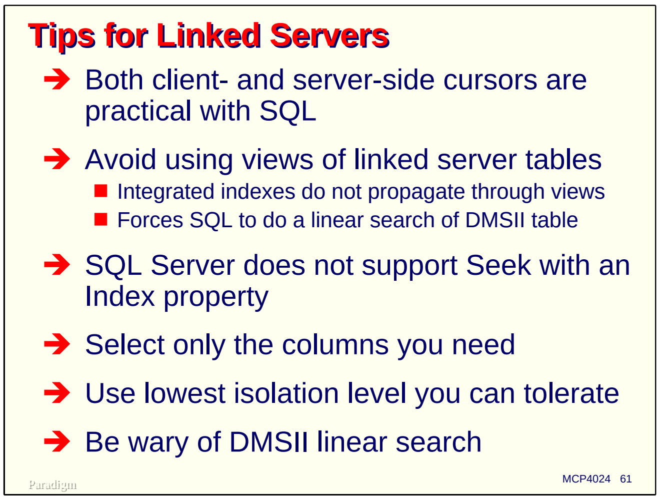
Distributed Query Example

```
select cat.cat_id, cat.unit, xcat.description

from ls1.sample.sample.catalog cat
   inner join extendedcatalog xcat
   on cat.cat_id = xcat.catalogid

where cat.category_id like "A%"
```

Paradigm MCP4024 60

This slide shows a very simple example of a distributed query. One table in the query is from a linked server to a DMSII data base. The other table is a native one from the SQL Server instance. Again, there is nothing special about the way that you write this query, other than using the four-part linked server name to reference the DMSII table.

Remember when doing distributed queries that the collating sequence for DMSII is EBCDIC. SQL Server handles this very well, but it can occasionally produce unexpected results.



Tips for Linked Servers

- Both client- and server-side cursors are practical with SQL
- Avoid using views of linked server tables
  - Integrated indexes do not propagate through views
  - Forces SQL to do a linear search of DMSII table
- SQL Server does not support Seek with an Index property
- Select only the columns you need
- Use lowest isolation level you can tolerate
- Be wary of DMSII linear search

Paradigm MCP4024 61

Here are a few tips involving linked servers and their use with DMSII data bases.

Unlike Direct Table Access, both client- and server-side cursors are practical with SQL Server. The reason is that SQL Server actually processes queries, and returns the results of the query as a rowset (or an ADO Recordset object). These results are typically much smaller than the entire table.

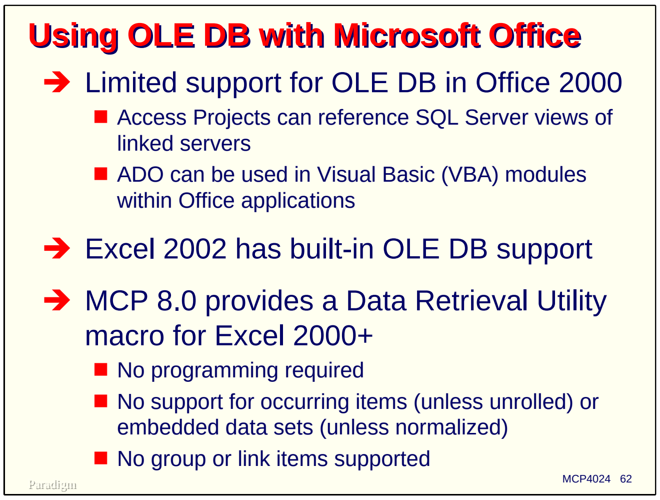
Avoid using views defined on linked servers as tables within queries. The reason for this is that integrated indexes do not propagate through views, and SQL Server will generally be reduced to sequential scans of the DMSII data sets in order to find records. This can lead to disastrous response times, especially when joins are involved.

You can use Direct Table Access through SQL Server, but the capability is limited. In particular, you cannot set the Index property for Recordset objects of SQL Server connections, so you cannot use the Seek method.

Select only the columns you need in a query. This will generally make the query run faster, and not as much data will need to be transferred over the network between systems.

Use the lowest isolation level you can tolerate. Levels above Read Uncommitted can require substantially more resources, especially if the result sets are large.

Finally, be wary of DMSII linear search when designing queries for use with linked server tables.



Using OLE DB with Microsoft Office

- Limited support for OLE DB in Office 2000
  - Access Projects can reference SQL Server views of linked servers
  - ADO can be used in Visual Basic (VBA) modules within Office applications
- Excel 2002 has built-in OLE DB support
- MCP 8.0 provides a Data Retrieval Utility macro for Excel 2000+
  - No programming required
  - No support for occurring items (unless unrolled) or embedded data sets (unless normalized)
  - No group or link items supported

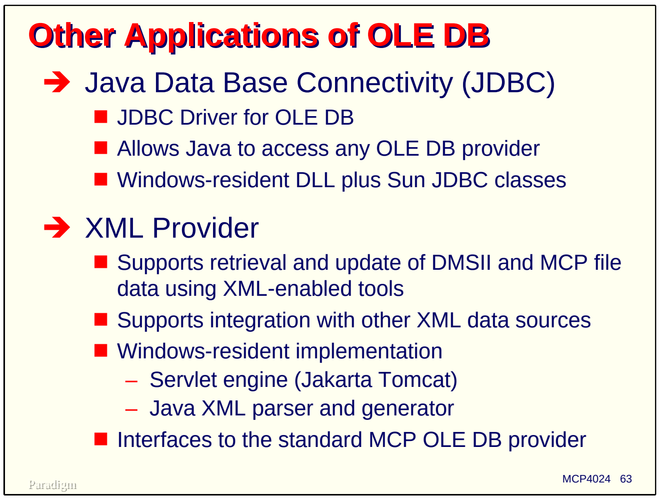
Paradigm MCP4024 62

The next application of OLE DB to discuss is Microsoft Office. This will be brief, because there's not a whole lot to say.

Microsoft Office 2000 was supposed to incorporate full support for OLE DB. It didn't happen. Only Access 2000 included support for OLE DB, and that was both limited and available only when interfacing to SQL Server. You can define views of linked server data bases and reference the views through Access "Projects," but since integrated indexes do not propagate through views, this capability is typically useful only for batch-style reporting. You can, however, use ADO in VBA modules for Access applications. This works the same as ADO in other environments and has good performance.

Excel 2002 has built-in support for OLE DB. You can retrieve data into Excel 2002 spreadsheets directly from the Unisys Provider.

MCP 8.0 provides a Data Retrieval Utility which can be used with Excel 2000 and higher. It is a macro which you plug in to your spreadsheet. It has some limitations (e.g., no support for chapters) but can be used without any programming expertise.



Other Applications of OLE DB

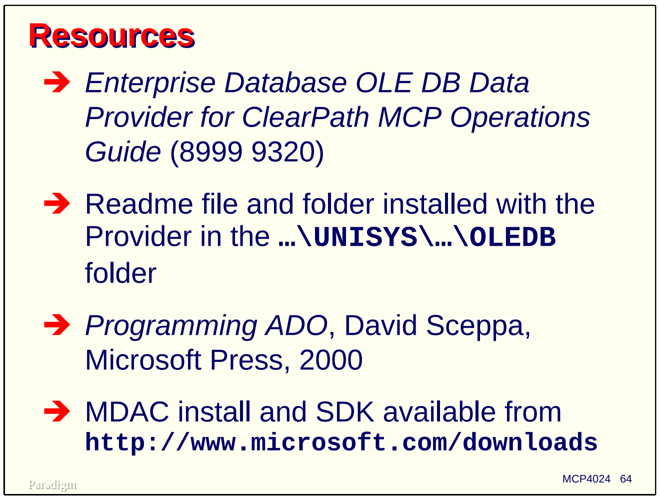
- Java Data Base Connectivity (JDBC)
  - JDBC Driver for OLE DB
  - Allows Java to access any OLE DB provider
  - Windows-resident DLL plus Sun JDBC classes
- XML Provider
  - Supports retrieval and update of DMSII and MCP file data using XML-enabled tools
  - Supports integration with other XML data sources
  - Windows-resident implementation
    - Servlet engine (Jakarta Tomcat)
    - Java XML parser and generator
  - Interfaces to the standard MCP OLE DB provider

Paradigm MCP4024 63

I'll also mention two other areas where the OLE DB Provider for DMSII is being employed.

The Java Data Base Connectivity (JDBC) implementation for DMSII uses OLE DB as the gateway to DMSII data bases. It employs a Windows-resident DLL along with the standard JDBC classes supplied by Sun. It can be used with any OLE DB provider, not just the one for DMSII.

The relatively new XML Provider also uses OLE DB as the gateway to DMSII data bases. It also supports integration of DMSII data with data from other XML-enabled data sources. The XML Provider is basically a Windows-resident implementation, comprising a servlet engine based on the Jakarta Tomcat web server and an XML parser and generator written in Java.



Resources

- *Enterprise Database OLE DB Data Provider for ClearPath MCP Operations Guide* (8999 9320)
- Readme file and folder installed with the Provider in the **`…\UNISYS\…\OLEDB`** folder
- *Programming ADO*, David Sceppa, Microsoft Press, 2000
- MDAC install and SDK available from **`http://www.microsoft.com/downloads`**

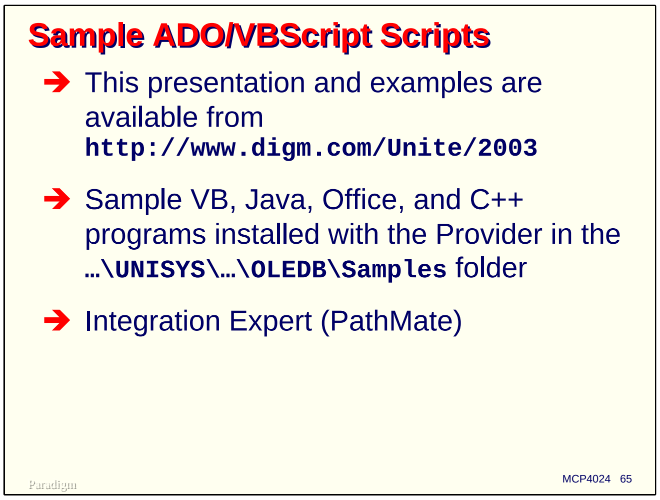
Paradigm MCP4024 64

The primary documentation resource for the OLE DB Provider is its *Operations Guide*. This is a PDF document. There are also a number of help files for the various Windows utilities used with the Provider.

When you install the Provider on a Windows system, a number of Readme documents are also loaded. There is a Readme.txt file that is loaded to the OLEDB root directory, and a Readme folder subordinate to the OLEDB directory. I strongly suggest that you look at all of this material, as it contains useful information not contained in the regular documentation.

The best book on ADO I have found is *Programming ADO* by David Sceppa. Unfortunately, this book is now out of print, although you may be able to find used copies through Ebay or Amazon. This author also has a book out on ADO.Net, so beware of which one you are getting.

The MDAC installation and an SDK are available at no charge from the Microsoft web site. The SDK includes complete documentation on OLE DB and ADO.



Sample ADO/VBScript Scripts

- This presentation and examples are available from `http://www.digm.com/Unite/2003`
- Sample VB, Java, Office, and C++ programs installed with the Provider in the `…\UNISYS\…\OLEDB\Samples` folder
- Integration Expert (PathMate)

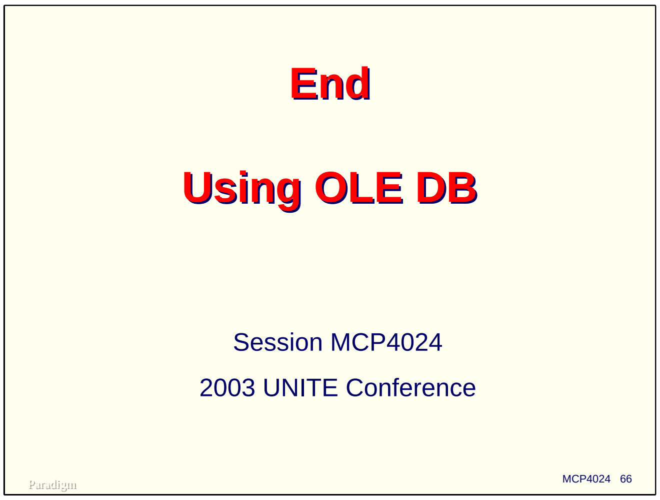
Paradigm

MCP4024 65

I have generated a number of small, sample VBScript files that use ADO to access DMSII data bases. There are also some Active Server Pages scripts which can be used with IIS. These are available, along with a copy of this presentation, from our web site at the URL shown on the slide.

There are a few sample programs installed with the Windows portion of the provider in a Samples folder under the root OLEDB directory. These contain a number of interesting comments that are useful if you are just getting started with OLE DB and ADO.

Finally, the Unisys Integration Expert has a number of scenarios which use OLE DB and ADO.



End

Using OLE DB

Session MCP4024

2003 UNITE Conference

Paradigm

MCP4024 66