Informix Dynamic Server 11.50 Fundamentals Exam 555 certification preparation, Part 1: IDS planning and installation

Skill Level: Intermediate

Jeffrey S. Bohm (jbohm@us.ibm.com)
Software Engineer
IBM

16 Jul 2009

This tutorial is the first in a series of tutorials designed to help you become familiar with all the different aspects of IBM® Informix® Dynamic Server (IDS) and help you get ready for the IDS Fundamentals Certification exam. This first tutorial covers the planning and installation of IDS—a great place to start to successfully understand and use IDS.

Section 1. Before you start

This tutorial teaches you about planning for an IDS installation. This planning consists of key questions that need to be answered, as well as where to get information about the product, its capabilities on your machine, and any pre-setup that might be needed.

After planning for the installation comes the actual installation. This tutorial talks about the different installation options available on each operating system and continues past installation into configuration of your first IDS instance, as well as how to start and stop that instance once it is configured.

With this knowledge you should be able to plan, install, and start an instance of Informix Dynamic Server that you can then use going forward with the rest of this series of tutorials. You should also be prepared to answer the types of questions asked in Section 1 of the certification exam.

About this series

This complimentary series of nine tutorials has been developed to help you prepare for the IBM Informix Dynamic Server 11.50 Fundamentals certification exam (555). This certification exam will test your knowledge of entry-level administration of IDS 11.50, including basic SQL (Structured Query Language), how to install IDS 11.50, how to create databases and database objects, security, transaction isolation, backup and recovery procedures, and data replication technologies and purposes. These tutorials provide a solid base for each section of the exam. However, you should not rely on these tutorials as your only preparation for the exam.

About this tutorial

This tutorial is dedicated to covering the topics in Section 1 of the exam, titled "Planning and Installing IDS".

Objectives

When you finish this tutorial, you should be able to perform the following:

- Explain the differences between IDS editions
- Describe different database applications
- Describe user setup and control, including role separation
- Understand the data types available in IDS, including both built-in and extensible types
- Explain and execute the different installation methods available on your platform
- Explain and execute the steps to configure an IDS instance
- Explain the different states of an IDS instance
- Execute commands to change the state of an IDS instance

Prerequisites

This tutorial is written for up-and-coming database administrators (DBAs). Although some basic database concept knowledge may help, it is not necessarily needed.

System requirements

To complete this tutorial, you do not need a copy of IDS. However, if you have one available to use, you will definitely get more out of the tutorial. If you don't have a copy already, you can download a free trial version (see Resources).

Section 2. Planning

First it was my parents, then my high-school counselor, my college professors, my business coach, and the list goes on; but they all essentially told me that same thing: In order to be successful, you need to have a plan. When it comes to successful database deployment, the same is true—you need to plan. Although you may cringe at the idea of having to take the time to create a plan, remember that it can end up saving you a lot of time later on.

So this first part of this tutorial covers the idea of planning. While planning, some of the questions you need to ask yourself include:

- What capabilities do I need in the database software?
- What type of applications am I expecting to connect?
- How many users are there, and where are those users connecting from?
- What type of data and how much data am I going to store?
- What is the expected response time from my application?
- Am I going to use existing hardware or buy new?
- Is one person going to maintain everything surrounding the data, or am I going to split out roles to a team?

The questions could go on, but let's stop there for now. The intent of this section is not to go in depth about all these questions but to get you thinking about them and discuss some high-level points about some of them.

IDS editions

Informix Dynamic Server comes in four different editions:

- Developer
- Express
- Workgroup
- Enterprise

The Developer Edition of IDS is a free product meant for application development and testing only. It contains most of the functionality of the Workgroup Edition but has no IBM technical support capabilities. It also has scalability limits for processors, memory, and storage.

The Express Edition is meant for small- to medium-sized businesses and is limited to running on Linux and Windows operating systems. It also contains most of the functionality of the Workgroup Edition with scalability limits.

The Workgroup Edition is meant for medium-sized companies or departmental servers of an enterprise. It is available on various Unix/Linux operating systems, as well as Windows and Mac OS X. Workgroup Edition adds additional functionality, such as limited Enterprise Replication (ER) and High-Availability Data Replication (HDR). The Workgroup Edition also has limits on scalability.

The Enterprise Edition contains all the functionality of Workgroup Edition with unlimited scalability. Enterprise includes full HDR and ER functionality, as well as additional functionality for the Continuous Availability Feature, Storage Optimization, LBAC, and more.

One of the first pieces of planning is deciding which edition of IDS is needed to support your business requirements.

Application types

The applications that connect to a database are usually split into two categories:

- Online Transaction Processing, better known as OLTP
- Decision Support Systems, better known as DSS, but sometimes also called data warehousing

"So what's the difference?" you ask. Let's look at an example.

OLTP

OLTP-type processing is like a call center application. When you call in to your credit card company (or help desk of any sort), the representative on the line usually asks you for something that is unique to your account, like an account number. That account number is used to search in the database to get a few records associated with only you. It could be just one record—maybe your account information; or it could be a couple dozen, like your account information plus the last 15 transactions on your account. Either way, it is designed to be a quick search that brings back the data in a matter of a couple seconds or less. Although it could bring back a couple dozen rows with data from several tables, the rows that it brings back are just a fraction of a fraction of the number of rows that are kept in the database. The 20 rows that the application may have brought back is therefore nothing compared to the 200 million rows in the table. This is not to make you feel small in the grand scheme of things; I'm only trying to point out a couple of key elements to an OLTP-type system. It:

- Returns a few rows from a query
- Has a very quick response time—usually a couple seconds or less
- Uses uniqueness and indices to get row set
- Is a read and write type environment
- Allows a lot of users

OLTP systems are designed to be read and write systems. User applications, like the credit card help desk, will not only select data from the database, but will also frequently update and insert new data.

DSS or data warehousing

DSS-type queries, for lack of a better term, are reporting-type queries. They are longer-running-, farther-reaching- (through the data) type queries. Taking the credit card company example from above, the CEO of the company wants to:

- Find out how many credit card transactions went through last month
- Have a report of all those customers that are delinquent on their accounts

Notice that these queries are not associated with one particular customer. The first query may use only one table to get its data (or it could use several), but it is having to do a much larger search over a span of data to get its results for the one month, and then aggregate (count) the data to get the final result. The second query is similar to the first, but brings back more data.

Many people think of DSS-type queries as queries that return a book of data. However, that is not at all true. Of the examples, the first query is a DSS query, but it only returns one row—one value, in fact—the count. The second query could return a book of data, depending on how many people are delinquent on their accounts.

The key elements associated with a DSS query are:

- Longer running queries; can be minutes to hours
- More resource intensive
- Accumulate more rows that match the criteria of the query
- Larger aggregation of the data (sum, min, max, count, and so on)
- Uses sequential scan on the table since larger data sets are involved.

DSS environments are more read-only-type environments. Although the data has to get into the system somehow, it is usually with bulk loads and then once if the system does not get updated or is updated very infrequently.

So don't categorize a query by the amount of data that it brings back, but instead by the key elements of the query itself.

The Data Model that is used during the design of the database is also very important. IDS can take advantage of three data models:

- Relational data model - A typical OLTP-type model
- Object-relational data model – Adds to the relational data model by using the extensibility options of IDS (extended data types, user-defined Routines, user-defined types, user-defined aggregates, and so on)
- Dimensional data model - typical DSS data model, supporting online analytical processing (OLAP)

Although in today's environments, a single database doesn't necessarily have the luxury of being just OLTP or DSS, it is always a good idea to try and figure out what the main application type for your database will be, and follow a design path that matches that type.

In order to accommodate the fact that many databases have both types of applications connecting to them at the same time, IDS has configurable parameters to help performance of both OLTP and DSS queries running on the same database.

Users

Another piece of planning is taking care of user requirements.

The basic questions to ask are:

- Do I need any special users or groups for the software?
- Where are the users connecting from?

- How many users do I expect to be connecting simultaneously?

The importance of answering these three questions is not only good for the planning of the database, but also for the planning of the hardware requirements.

IDS on Unix/Linux requires an "informix" user account as well as an "informix" group set up on the installation machine. Windows loosens that requirement and allows for either an "informix" user or the use of a local system user and an "Informix-Admin" group. During installation on Windows, you can choose which user account you would like to use.

By default, the user "informix" is considered the "super-user" of the IDS software. It has access to everything and can do anything that needs to be done with the IDS software.

IDS 11.50 uses external user authentication, which just means that it doesn't have users set up inside the software, but relies on other software to help it authenticate a user. This external software can be the operating system (OS) authentication mechanism of the machine where IDS is running, LDAP, MS Active Directory, or a Pluggable Authentication Module (PAM).

Note: Whichever way you decide authentication, the important thing to remember is that ALL users that need access to the database have to have an account and password with an external authentication mechanism so the database can authenticate them and allow them access to the data. This can take some major planning depending on whether you have two user accounts connecting or 2000 user accounts connecting.

As mentioned previously, by default, the "informix" user is the "super-user" of the IDS installation. However, certain specifications require the administrative power to be split out under multiple users, with each user having a subset of the total administration. This is known as role separation. Role separation is meant to be a kind of checks-and-balances mechanism. IDS role separation splits this out under two categories:

- Administrative tasks for people who run the instance
- Audit tasks for people who audit what is happening on the instance

Role separation must be turned on during the installation of the software by either setting the `INF_ROLE_SEP` environment variable before starting the installation process, or by actively turning it on during the interactive installation process. Role separation can only be turned back off by uninstalling/reinstalling the IDS software. Role separation uses user accounts on Windows and group accounts on Unix/Linux. If role separation has been turned on, the installation process will ask for the information needed for these user and group accounts.

The three users who take part in role separation are:

- Database System Administrator (DBSA) - controls general operations of the instance
- Database System Security Office (DBSSO) - determines what to audit
- Audit Analysis Officer (AAO) - monitors the audit trail

Other users that aren't part of role separation but might need some planning for include:

- Database Administrator (DBA) - manages a database on the instance
- Operating-System Administrator (OSA) - satisfies OS requirements
- System Users - any user that needs to get to the data

Data types

Part of database planning is being familiar with what data types are available for use to store the data. The easiest way to describe how data is stored in a database is by using a spreadsheet analogy. By now, you hopefully have used or seen some type of spreadsheet program (for example, Microsoft® Excel®). When you look at the spreadsheet, it is organized in rows and columns of cells. When you look at a column of cells (vertical data), usually all data in the same column is alike—a column of dates, or a column of money values—with a column header that describes what the data means.

When you look at a row of cells (horizontal data), all the data in the cells together describes one particular instance of what the spreadsheet is representing. In the example below (Table 1), this spreadsheet describes customers of the XYZ Shoe Store. Each row describes info about one particular customer, where each column describes one concept being kept about all the customers. So for each customer, you are storing his name, age, shoe size, the last time he bought shoes from the store, and how much he bought during that transaction.

Table 1. Data for examples

Name	Age	Shoesize	Last Trans Date	Last Trans Amt
Fred Flintstone	29	11.5	3/29/2009	$79.35
Barney Rubble	31	9	1/15/2009	$103.75
Dino Flintstone	7	16	11/26/2008	$129.95

Databases don't keep data in spreadsheets, but instead in tables. Tables, just like spreadsheets, are made up of rows and columns. During definition of the table you specify what columns of data are being held and what data type each column uses. A data type constrains the type of data the column can hold. If you define a column to be of an integer data type, then that column cannot contain letters; it is constrained to only containing whole numbers. The rows of the table are added as your applications start storing data in the columns of the table. Data is stored one row at a time. The example in Table 1 above has five columns—the first column, Name, is of character/string data type; Age is of integer data type; Shoesize is of decimal data type; LastTransDate is of date data type; and LastTransAmt is of money data type.

Now that you have an idea of where data types are used, let's take a look at the data types that are available in IDS.

Available data types

- Built-in data types
  - BIGINT - whole numbers from -(2^{63}-1) to 2^{63}-1
  - BIGSERIAL - whole numbers from -(2^{63}-1) to 2^{63}-1, automatically incremented by the server; sometimes used as a surrogate primary key
  - BLOB - Binary Large Object up to 4TB in size, storing objects in their native format
  - BOOLEAN - 't' or 'f' value; can be tested in expressions
  - BYTE - older version of BLOB with 2^{31} byte theoretical size limit and a practical limit determined by your disk capacity
  - CHAR(n) - stores 'n' characters of data; if value is greater than n, blank padded up to size 'n'
  - CHARACTERVARYING(m,r) - ANSI-compliant VARCHAR
  - CLOB - Character Large Object up to 4TB in size; stores character data
  - DATE - the calendar date; default format MM/DD/YYYY; can be changed with `DBDATE` environment variable; specifies a point in time.
  - DATETIME - the calendar date and time; default format YYYY-MM-DD HH:MM:SS.FFF; can be changed with `DBTIME` environment variable; specifies a point in time.
  - DECIMAL(p,s) - decimal values, where 'p' is the total number of digits and 's' is the number of digits to the right of the decimal point

- NUMERIC(p,s) - same as DECIMAL
- DOUBLE PRECISION - same as FLOAT
- FLOAT - double-precision floating-point number with up to 17 significant digits
- IDSSECURITYLABEL - a VARCHAR(128) used only with Label Based Access Control (LBAC)
- INTEGER - whole numbers from $-(2^{31} -1)$ to $2^{31} -1$
- INT8 - whole numbers from $–(2^{63} -1)$ to $2^{63} -1$
- INTERVAL - format same as DATETIME, but specifies a span of time
- LVARCHAR(m) - long variable character length field with 'm' as max size; only use as much space as needed to store data up to 'm' size; max size 2GB when used with UDT, otherwise max 32K
- MONEY(p,s) - just like DECIMAL value, except formatted with money characters; default is $ and ., but can be changed with `DBMONEY` environment variable
- NCHAR(n) - stores fixed length character data, but includes the use of Global Language Support (GLS) to store both single-byte and multi-byte character sets that are supported by the database locale; also allows for the use of localized collation sequences
- NVARCHAR(m,r) - same as VARCHAR, but with special characteristics like NCHAR
- REAL - same as SMALLFLOAT
- SMALLFLOAT - single-precision floating point numbers with approximately nine significant digits
- SERIAL - whole numbers $–(2^{31} -1)$ to $2^{31}-1$, automatically incremented by server; sometimes used as primary key
- SERIAL8 - same as SERIAL with a range of $–(2^{63} -1)$ to $2^{63}-1$
- SMALLINT - whole numbers from -32767 to 32767
- TEXT - older version of CLOB with maximum size of 2GB
- VARCHAR(m,r) - variable character length field with 'm' as the maximum size (up to 255) and 'r' as the smallest reserved space; if value being stored is less than 'r', the value will be space padded up to size 'r'; if value being stored is greater than 'r' and less than 'm', then it will only use as much space as needed to store the value

- Extended data types
    - Complex data type
        - Row
        - Collection
    - User-defined data types
        - Distinct types
        - Opaque types

The built-in data types are considered atomic, which means that they can't be broken into smaller pieces. Each built-in data type has its own characteristics that make it unique.

Listing 1 provides an example of using built-in types when creating the customer table (data from Table 1):

Listing 1. Using built-in types

```
CREATE TABLE customer
        ( Name CHAR(55),
                Age INTEGER,
                Shoesize DECIMAL(3,1),
                LastTransDate DATE,
                LastTransAmt MONEY(5,2)
        );
```

Data type extensibility

Since the built-in data types cannot encompass every possible type of data that users want to store, IDS has the capability to extend them by combining the built-ins together or by adding new user-defined data types. The complex data type category is made up of two data types known as row and collection.

The best way to describe a ROW data type is that it mimics a row in a table, but you put that whole row into a single column. So it is a multi-part data type made up of the built-in data types. It is no longer atomic. Row data types can be named or unnamed.

Example:
Using the XYZ Shoe Store example from above, maybe you want to store the full name of the customer—first, middle, and last. Instead of creating three different columns—one for each part—you can create a single column that has three parts.

The COLLECTION data type is actually a category of three underlying data types called set, multiset, and list. The collection data type allows groups of data, all of the same data type, to be stored together in a single column.

Example:
Using the XYZ Shoe Store example from above, maybe you want to store the customers' favorite brands of shoes. You don't want to create multiple columns because you because you might not know how many columns you would need. Some people have only one favorite brand; other people have five favorite brands when it comes to shoes. Since all brand names are just character strings, they are all of the same data type. So you can create a column called FavBrands of SET type that stores character string data. Now that single column can store as many brands as that customer has favorites. With SQL, you can then go in and select all of the customers whose FavBrands include Nike regardless of how many favorite brands are listed. You would select a SET type since SET doesn't allow duplicates in the collection. It wouldn't make sense to have a value of 'Nike','Keds','Nike'. Both multi-set and list allow duplicates. Collections do not allow Null elements, so when defining a collection the NOT NULL constraint must be specified.

Listing 2 provides an example for creating the customer table using the above ROW and COLLECTION ideas:
Listing 2. Using ROW and COLLECTION

```
CREATE TABLE customer
        ( Name ROW( fname CHAR(15), mi CHAR(1), lname CHAR(35)),
                Age INTEGER,
                Shoesize DECIMAL(3,1),
                LastTransDate DATE,
                LastTransAmt MONEY(5,2)
                FavBrands SET(CHAR(10) NOT NULL)
        );
```

After making the changes my table data would look something like:

Table 2. Data being used for examples

Name			Age	Shoesiz	Last Trans Date	Last Trans Amt	FavBrands
FName	MI	LName					
Fred	F	Flintstone	29	11.5	3/29/2009	$79.35	Rockers, Nike
Barney		Rubble	31	9	1/15/2009	$103.75	Treadwear
Dino	T	Flintstone	7	16	11/26/2008	$129.95	Doggear

User-defined types

Although the built-in types and complex types can cover much of the data that a user would want to store, it is possible that new data types will be needed. New applications and technologies bring new things that could need new data types. So, instead of IDS deciding when a good time to add a new data type is, IDS has empowered you with the capability to add new data types whenever you need them. These data types are known as user-defined types, or UDTs. You design it, you tell IDS how to interact with it, and there you go—you have great new functionality.

If only everything in life were that easy, including this, then we could all be doing it. Actually, it isn't necessarily hard; there are just several things that you have to do to make it all work. More of this will come up as this tutorial continues to discuss UDTs and support functions for UDTs. So back to UDTs….

As seen in the available data types list above, the first type of UDTs you have is called a distinct type. To put it simply, a distinct type is just the renaming of a built-in data type. It inherits the basic characteristics of the built-in type, but not the support functions.

Example: Create distinct type Shoesize as DECIMAL(3,1):

Since Shoesize is being defined as a decimal, it will take on the storage characteristics of the decimal type. However, not everything you can do with a decimal value makes sense to do with a Shoesize value. An example would be addition. It doesn't make sense to add two shoesizes together to make another shoesize. The addition function (+) is considered a support function in IDS. Since IDS doesn't know whether the support functions will make sense to the newly defined distinct type, it just automatically disables all support functions on the distinct type. Because of this, you also can't compare a shoesize value to a decimal value. Comparing values is also considered part of support functionality. Even though shoesize is defined as a decimal, they are considered two different (distinct) types that can't be compared. If you want to compare a decimal value to a shoesize value you have to first cast one of them to be of the same type as the other, and then the comparison can be made.

So you ask, "If no support functions are supported on this new 'distinct' data type, what good does it do me?" Well, you might have something special you can do with shoesizes, like telling if one is larger than another one. This is kind of a bad example, since shoe sizing follows a normal incremental value just like decimal values do. However, say that you made this to incorporate XS, S, M, L, XL, and XXL values as well. You would have to change shoesize from a decimal to a character type. But now if you wanted to see if a customer's shoe size had grown or shrunk since he last bought a pair of shoes, you could no longer rely on normal ordering (alphabetical in this case). If you tried to rely on alphabetical ordering, then the shoe sizing chart would look like L, M, S, XL, XS, XXL. But that isn't the case because you know that XS is the smallest in the shoe sizing list. Since you couldn't rely on IDS's built-in functions to help with this, you need an alternative. IDS's alternative is to allow the writing of external routines that can then be called in normal SQL. So you could write a routine to do the shoe sizing comparison for you. These new routines are known as user-defined routines (UDRs). This tutorial covers UDRs a little later on.

The other kind of UDT is known as an OPAQUE data type. Just like the name describes, an opaque type is a new type that IDS doesn't understand at all. Not only does the data type have to be described to IDS, something like a C structure, but you also have to tell IDS how the structure should look like on disk, how to convert it between disk format and display format, how to index it, and any support functions you want to use for it. People always ask, "If I have to do so much work, why should I use IDS when I can just write my own application?" Although OPAQUE data types can take a lot of work to set up, they can have high reward because you automatically have access to all the rest of the database capabilities. Once you define the support functions, IDS can use those anywhere in normal SQL statements, making it look just like any built-in type. So now you have transactional integrity, backup and restore, storage, user functionality, integration with other data types, and everything else that is already built into the Relational Database Management System (RDBMS).

A prime example of using OPAQUE types is the GeoSpatial Datablade that defines a GeoPoint data type, which stores four-dimensional geographic coordinates. The support functions around this help analyze distance, proximity, intersection, and other functions based on geographic points. These functions can be included inside of normal SQL statements so that character data, geopoint data, and any other data type data can come out in the same result set.

User-defined routines

The support functions for any UDT can be written in C, Java, or Stored Procedure Language (SPL). In fact, it doesn't just stop with UDTs. A DBA can create a UDR to support built-in data types as well.

Example:
The average (AVG) aggregation function (routine) for built-in numeric data types completely ignores NULL values when it is doing its calculations. So, maybe for your analysis you need an AVG routine that can take NULL values and convert them to zeros and include them in the calculation. You can write a UDR that overloads the built-in AVG function and use your new routine for built-in data types, UDTs, or both.

Notice in the example, the terms function and routine are synonymous. This is because the term routine is just a category that encompasses two actual elements: functions and procedures. So when you create a routine you actually use either the `CREATE FUNCTION` or `CREATE PROCEDURE` SQL statement.

"Why the two?" you ask. By ANSI standards definition, a function can both take and return values, where a procedure can take values but shouldn't return anything. An example of a function would be the SUM function—you pass it 2 and 2, and it returns 4. An example of a procedure would be updating a person's salary—you pass it the new salary but it doesn't need to return anything back to the user. As of 11.50, IDS is not completely ANSI-compliant because it still allows return values from procedures. This is done for backwards compatibility. Just make sure you keep the ANSI standards in mind when writing any future routines.

Datablades

Before leaving the extensibility topic of UDTs and UDRs, let's look at one more concept known as a datablade. Since we have discussed what UDTs and UDRs are, the Datablade concept should be an easy one. A datablade is the packaging together of UDTs and UDRs for a specific reason. The reason is usually because they have something in common, like a given functionality. An example would be the Geodetic Datablade. Inside the Geodetic Datablade are all the UDTs and UDRs needed to do geospatial storage, retrieval, indexing, and all kinds of analysis.

Release notes

Let's pretend for a second that you need a new printer for your computer. You head to the local computer peripheral store, pick one out you like, and take it home with you. When you get home, you open the box, and the first thing you do is pull out the User Manual, read it cover to cover, and then unpack the rest. Right? If you are like the other 99% of us, you may pull out the manual, but you set it aside because you are too excited about getting your printer hooked up. Well, peripheral manufacturers figured that out, and to help out, they put something else in the very top of the box—a thing they call a "Read Me First" paper with the 10 or 20 easy steps to getting the printer working.

IDS has incorporated this same idea in two parts. The first is a README.html file that is in the top directory when you extract the software from the installation media. The README file has very basic installation instructions as well as links to several other documentation files known as "release notes". (This tutorial discusses more about installation in the "Installation" section.) The release notes describe new features, supported releases, kernel parameter values, deprecated features, and information about known problems and their workarounds. If you are upgrading from one release of IDS to another, the release notes are a great place to learn about changes made between releases. If you are installing brand new, they are still a great place to find out information about this version and other locations for information. If you are inheriting an already installed product, the release notes can still be found in a sub-directory of the directory where IDS is installed. Let's call the directory where IDS is installed INFORMIXDIR. So the release notes would be found in /INFORMIXDIR/release on Unix/Linux or C:\INFORMIXDIR\release on Windows. Actually, they would be found in a localized set of directories under that. These localized directories follow a Global Language Support (GLS) convention. If you are installing the US English version of the product on Unix/Linux, the total directory structure will look like: /INFORMIXDIR/release/en_us/0333.

Depending on your country and language, the last two sub-directories could be named differently but will still have a similar structure. Inside that final directory you will find release notes, in both html and text format, for all the IBM Informix products installed on this machine.

Locales

The last part of the release notes section started bringing in the idea of Global Language Support (GLS). Not all countries of the world speak the same language, have the same alphabet, or use the same money. So you shouldn't expect the software to only be in one language either. That is where localization comes in. "Localization" is the process of transforming a product to meet a specific cultural environment. As part of localization, you create culture-specific resource files, translation files for messages and errors, translation files for the product user interface, and set the date, time, and money formats.

Defining a locale takes it one step further; it uses the localized files, but also specifies a code set (character mapping) and collation sequence (dictionary sort order). This separation allows multiple locales to exist for the same localization, similar to a country that has multiple regions. All regions speak the same language, but each region has their own slight variation to it. The product and hardware type will determine the default locale that is used when creating a database. This is known as the DB_LOCALE. For IDS product bought in the U.S. and installed on Unix/Linux, the default DB_LOCALE is en_us.8859-1 (also known as en_us.819). If installed on Windows, the default DB_LOCALE is en_us.1252. If you want to change the DB_LOCALE from the default, it has to be specified at database creation time by setting the `DB_LOCALE` environment variable in the session that runs the `CREATE DATABASE` SQL statement.

ISO 8859-1 character encoding is for the Latin alphabet, which many countries share.

Although the DB_LOCALE specifies the default localization for the database, clients that connect to the database have the capability to use a different localization. Clients have the CLIENT_LOCALE environment variable that specifies the locale of the client. In order for the database and client to exchange information, their locales have to either be the same or compatible (convertible).

When storing character data for GLS, it is important to use the NCHAR and NVARCHAR data types, instead of the CHAR and VARCHAR data types. The "N" data types allow the collation (sort order) of data based upon the CLIENT_LOCALE, instead of just on the DB_LOCALE. The `SET COLLATION` SQL statement allows for a client to change this dynamically inside their current session.

Listing 3 shows the formatting concepts of GLS locales. Since a lot of countries use the Latin alphabet, many of the code sets for these countries are compatible. After creating th Shoe store table from above (see Table 1) and populating it with one row, I ran the following `SELECT` SQL statement multiple times, but changed the `CLIENT_LOCALE` environment variable to something different but still compatible with the DB_LOCALE for each run.

Listing 3. GLS locals

```
SELECT name.fname, SUBSTR(TO_CHAR(lasttransdate,"%B %d %Y"),1,15), lasttransamt
FROM customer;

$ export CLIENT_LOCALE=en_us.8859-1         --(setting for US English)

fname           (expression)      lasttransamt

Fred           March 21 2009            $85.43

$ export CLIENT_LOCALE=nl_nl.8859-1         --(setting for Dutch Netherlands)

fname           (expression)      lasttransamt

Fred            maart 21 2009           F 85,43

$ export CLIENT_LOCALE=de_de.ASCII         --(setting for Germany German)

fname           (expression)       lasttransamt

Fred            Mrz 21 2009            85,43DM

$ export CLIENT_LOCALE=pt_br.8859-1         --(setting for Brazil Portuguese)

fname           (expression)        lasttransamt

Fred            Março 21 2009         (85,43)R$
```

Notice how by just changing the CLIENT_LOCALE, the data format comes out differently. (the format, not the data). You have to be careful when you use this because a date is a date in any country, but $85 US is not equal to 85 Brazilian Reals. Additionally, the reason why all the dates came out in Month Day Year format is because of the call to the `TO_CHAR` function with the formatting set to %B %d %Y.

Listing 4 shows the output if you remove the call to the `TO_CHAR` function and rerun the SQL statement with the given CLIENT_LOCALE:

Listing 4. Output

```
$ export CLIENT_LOCALE=pt_br.8859-1

fname           (expression)         lasttransamt
```

```
Fred                  21/03/09               (85,43)R$
```

To recap, GLS libraries allow an application to meet cultural expectations of the data without having to change the application. GLS locales, through the use of special data types, environment variables, and SQL statements, allow for the dynamic formatting and sorting of data to meet the clients standards. All of this can be done without having to write the client application differently for every country/region of the world that it might be used in.

Summary

In the first part of this tutorial, you have learned about:

- The two different types of applications used against RDBMS systems
- How user authentication is treated in IDS
- All the different data types that IDS has available to use
- IDS extensibility through UDTs and UDRs
- IDS "Read-me first" files and release notes
- GLS localization

If you want more information on planning for a database, take a look through the "Installing IDS" topic in the IBM Informix Dynamic Server 11.50 Information Center (see Resources).

Section 3. Installation

After planning for IDS, the next step is to actually install the product. IDS is available on Unix/Linux, Windows, and Mac OS X. The installation choices available to you depends on which operating system you have.

Unix/Linux

IDS installation usually comes in two parts:

- Extract the product from the package media (CD, tape, or download)

- Run the installation script

Copying the product from the media is usually accomplished with the tar `–xf` command. The product should be un-tared into the directory location where the product will be installed.

Since the IDS product suite encompasses more than just one product, the installation process can be done all at once or in steps at the product level. By running the `ids_install` command, the installation script will install the IDS server as well as any other related products that are in the same directory. The `installserver` command just installs the IDS server and ignores other products.

Root permissions are needed to run the installation script.

Once logged in as root, the installation can be accomplished in multiple ways:

- Console mode (default): This mode uses the default terminal in plain text type and asks for responses from the installer on license acceptance, installation location, installation mode, role separation, and demo server initialization. Once these questions have been answered and the summary has been approved, the installation will take place.
  The installation mode option consists of either typical or custom installation.
  Some of the custom installable features include:
  - Extensibility features
  - GLS
  - Backup and restore utilities
  - Enterprise replication
  - Data-loading utilities

  The demo server initialization creates, sets up, and initializes an instance of IDS based upon a provided configuration file or the default configuration file.
  Example of installation command:
  ```
  install_ids
  ```
- GUI mode: This mode is used when the `–gui` option is specified with the install command. The GUI installation is just like the console installation but uses a Java graphical interface for interaction with the installer.
  Example of installation command:
  ```
  installserver -gui
  ```
- Silent mode: This mode allows for non-interactive installation. Silent mode uses an .ini file for the response information that would normally come

from the keyboard or mouse in the console and GUI modes. IDS comes with two default .ini files that can be used, or you, the installer, can create your own. An .ini file can be created automatically during an interactive installation by specifying the `-record < filename >` option on the installation command.
Example:
`installserver -record myresponses.ini`
To use the custom created .ini file, you must specify the `-options < filename >` option on the installation command.
Example:
`install_ids -silent -options myresponses.ini`
If either of the default .ini files is used (bundle.ini or server.ini) the `–acceptlicense=yes` option must be specified on the installation command; otherwise, the installation will not succeed.
Example:
`installserver -silent -acceptlicense=yes`

Some of the other options that can also be specified during installation include:

- `-javahome <javadir>` to use an already installed JRE
- `-P installLocation=<installdir>` to specify a different installation directory
- `-log <logfilename>` to specify a non-default log filename

It is possible to have multiple versions of IDS installed on the same system at the same time. The only thing that is required is that they are installed in different directories. The environment variable `INFORMIXDIR` points to the product directory that should be used when starting an IDS instance.

Windows

IDS installation on Windows is either done in graphical mode or in silent mode.

If the installation media comes from a download file, you must extract the files into a folder structure using the appropriate tool based upon the file type. If the installation media is a CD, you can launch the installation straight from the CD.

To start the installation, either run Launch.exe or setup.exe.

Launch.exe will start the GUI installation process that will take you through the following steps:

- Selecting the products to install

- Accepting the license agreement
- Selecting the installation mode (typical or custom)
- Setting up the informix user account and password, if one doesn't exist
- Selecting the installation directory
- Agreeing to the summary info

Installation mode of typical or custom is similar to that which was described in the Unix/Linux section above, except it also allows for the following:

- Specifying a different user account than a local 'informix' user to run under
- Enabling of role separation
- Creating a demo server with or without initializing it
- Starting of the Instance Configuration Wizard
- Starting of the ClusterIT utility

To install on Windows in silent mode, the setup.exe command is used in a command-line environment. Just as described in the Unix/Linux environment, silent mode uses an .ini file for the response information during installation. Either the default server.ini file can be used, or a custom file can be created.

The following is an example of the installation command when using the default server.ini file:

```
setup.exe –s –f1"C:\IIF\server.ini"
```

The way to use a custom .ini file during installation is to change the name of the file in the `–f1` option as shown in the above command example.

To create a custom .ini file from the responses given during a GUI install, you must first run the command `setup.exe –r –fl"C:\temp\mysilent.ini"` from a command prompt before starting the GUI install.

Of course you can change the path and filename to suit your environment.

Once the installation is complete, the new .ini file is ready to use for all future silent installs with the same configuration.

Windows installation creates a log file in $INFORMIXDIR%\logs\, which shows all the installation activity. The log file location can be changed with the `–f2"<filename>"` option when using the silent install.

Windows allows for multiple versions of IDS to be installed on a computer with the `-multiple` option.

Mac OS X

The Mac OS X installation is similar to the Unix/Linux install. It has the GUI and the unattended (silent) installation methods. Root privileges are required to do an unattended install.

The GUI method is started by opening the iif package and entering the system administrator password when prompted. The installation proceeds with prompts for:

- informix user account info, if it does not exist
- License acceptance
- Installation directory
- Products to install
- Installation mode (typical or custom)
- Role separation
- Demo database server creation
- Automatic kernel tuning
- Summary approval

The custom installation mode allows you to select which features are installed, as described in the Unix/Linux installation section.

Unattended, or silent, installation is done similar to the silent installation on Unix/Linux. On Mac OS X, the bundle.ini file must be used and customized to meet your installation needs.

Not: Before installing, make sure to change the `-G licenseaccepted=false` to true in the .ini file.

Once the bundle.ini file is customized, open a terminal window, change to the directory where the iif file exists, and, as root, run the following command:

```
Installer -pkg iif.11.50.FC#.macosx64.pkg -target /
```

Make sure to substitute the appropriate number for the # (number) sign in the above command.

Secure installation path

During the installation of IDS 11.50, there is an option to automatically secure the installation path. Securing the installation path verifies that the directories in the installation path have secure owners, groups, and permissions set appropriately for the database server.

If you choose not to secure the path at installation time, you can do it manually at any time.

Note: IDS will not initialize if the installation path is not secure.

To secure the path at a later date, do the following:

1. Run `$INFORMIXDIR/bin/onsecurity –r $INFORMIXDIR` to create a shell script.
2. Run `$INFORMIXDIR/tmp/secure.sh` to secure the path.

Securing the path through this manual process allows a system administrator to see what permissions are going to be assigned to any directories in the installation path and take appropriate action for any extra directories that might exist in the path, but not be part of the IDS installation.

Configuration

Once the product is installed, it is time for the next step, which consists of configuring the operating system, the environment that IDS runs in, and the IDS server instance itself.

Here are a couple of definitions you should know before you start:

- IDS instance - A defined set of operating system resources available for use by one or more databases. Sometimes an instance is also referred to as a database server or a database engine. This set of resources consists of disk space, processes, and memory.
- Relational database - A collection of data organized into tables for rapid search, retrieval, and storage.

So, as you are reading through the rest of this tutorial, keep in mind that Instance, engine, and database server are synonymous.

OS

Since configuring the OS is usually a System Administrator (SA) function, you might need the help of the SA to complete this task.

IDS ships with a file called machine notes. This file exists in the same location as the Release Notes that were described in the "Release notes" section. The machine notes file consists of recommendations for the OS kernel configuration parameters that are appropriate for the machine type where IDS is installed. The tunable configuration parameters depend directly on the OS manufacturer.

On Unix/Linux, the most important kernel parameters to tune are shared memory, semaphores, files, and users. Special I/O might also need to be tuned in the kernel.

Listing 5 shows an example from the machine notes for an HP-UX OS:
Listing 5. Machine notes for an HP-UX OS

```
On HP-UX, these entries are in /stand/system.

    nfile: 5000
    semmni: 3000
    semmns: 3000
    semmnu: 1400
    shmmax: 4294967296
    shmmni: 500
    shmseg: 500
```

Note: The values given in the machine notes are only recommended values based on testing of the product. If the values differ greatly from the values already set up in your kernel, make sure to seek advise from your system administrator on how changes to these values could affect the system.

On Windows, the most important configuration parameter is for memory. The capability to access more than the default amount of memory address space has to be enabled in the boot.ini file. By changing this value, you can increase this from approximately 2GB to approximately 3GB. Although not a tunable parameter, it is important to note that all data in a Windows IDS instance must be stored on NTFS partitions, physical drives, or logical disk partitions.

Environment

IDS relies heavily on the environment that it is started up in. Because of this, the environment variables that set up that environment are very important to understand and set correctly. There are five main environment variables for the IDS instance:

- `INFORMIXDIR` – The full path to the directory location of the installed product.
- `INFORMIXSERVER` – The name of the instance to be started.

- `PATH` (optional) - Should include $INFORMIXDIR/bin for convenience.
- `ONCONFIG` (optional) - Name of "all-important" configuration file.
- `INFORMIXSQLHOSTS` (optional) - Points to connectivity information file.

As you may notice, only two of the five environment variables are required; the other three are optional. This tutorial describes why this is true in more detail.

The location and command to set environment variable depends directly on the OS being used.

An example using Unix and Korn shell:
```
export INFORMIXDIR=/usr/informix
```

Because Windows has multiple places where environment variables can be set, the following rules of precedence apply:

- Setting in Setnet 32 application
- Setting on command line before running application
- Setting in Windows as a user variable
- Setting in Windows as a system variable
- Default value

The `INFORMIXDIR` environment variable points to the location where the product is installed. This is important because this path is prepended to some of the values that are used inside of the IDS executable. Without this set, IDS wouldn't know where to look for certain files that are needed to run successfully.

The `INFORMIXSERVER` environment variable corresponds to the name of the IDS instance that the environment will be pointed to by default. This environment variable is important to every client connection that tries to access the IDS instance, whether that client is internal or external. An internal client would be a utility that comes with the IDS software. An external client would be any application that uses SQL to talk to the database. "How do you name the IDS instance?" you ask. That is discussed a little later in the "Configuration file" section.

The `PATH` environment variable should be changed to include $INFORMIXDIR/bin. Although this is optional, it can be very convenient. It is much easier to type `oninit` than to have to type `/usr/informix/bin/oninit` (assuming /usr/informix is where the IDS product is installed).

The `ONCONFIG` environment variable is set to the name of the configuration file to be used by the IDS instance. Every instance has only one configuration file that it uses at any given time. It is possible to use a different configuration file, but this takes a stopping of the software, changing the `ONCONFIG` environment variable to point to a different file, and then restarting the software.

Note: The `ONCONFIG` environment variable is set to just the name of the file. It is *not* set to the path of the file. Example:

```
export ONCONFIG=onconfig.production
```

The onconfig file *must* reside in $INFORMIXDIR/etc, so you don't need to know where the file exists, just the name of the file to use in that directory. The onconfig file can be named anything you want it to be; however, the standard has become to name it onconfig.something, replacing the "something" with a meaningful name, as in the above example ("production"). It is also possible to use a file named onconfig, if that is your decision. Then the `ONCONFIG` environment variable becomes optional. The `ONCONFIG` environment variable is only important for DBA-type work (for example, starting and stopping the instance). Normal SQL client activity does not need to have the `ONCONFIG` environment variable set.

The `INFORMIXSQLHOSTS` environment variable is set to the *full pathname and filename* of the file being used for connectivity information. Example:

```
export INFORMIXSQLHOSTS=/work/jbohm/mysqlhosts
```

This parameter is optional because if it isn't set, IDS looks for the file $INFORMIXDIR/etc/sqlhosts to have the information that it needs. "What information is that?" you ask. This tutorial describes it in the "SQLHOSTS" section.

It is important to note that every client connection, internal or external, requires connectivity information. So every client either expects to get its information from the default file or from the file pointed to by the `INFORMIXSQLHOSTS` environment variable.

The five environment variables listed above are not the only ones available for use with IDS. In fact, IDS easily has over 100 more environment variables that can be used to control different aspects of the software. This tutorial has already named a couple earlier—DB_LOCALE and CLIENT_LOCALE that control GLS settings. Just remember that the five listed above are the most important, and two of them are required.

SQLHOSTS

The sqlhosts file is needed for connectivity information. It is a vital piece of setting up a database because it is used by every client, internal or external, that connects to the instance. If not set up correctly, nobody will be able to connect to the database and get data.

IDS is made to run locally or remotely (distributed environment) from the client. So, in order for the client to successfully connect to IDS, it needs to know where the IDS instance resides and how to get to it. Think of the sqlhosts file as the phone book of IDS. It is a listing of all the IDS instances available (by name), where they reside (hostname or IP of computer), and what service port to use when sending a request, just like the regular phone book lists people by name, where they live (address), and how to get in touch with them(phone number). The service port part of the sqlhosts specifies what port IDS accepts incoming SQL statements from its clients. This is the same as the way that telnet uses port 23 or http uses port 80, by default, to accept incoming requests. The port number used by IDS is not a global standard, but instead is configured by the DBA in the sqlhosts file.

The general form of the sqlhosts file is five columns and looks like Table 3:

Table 3. SQLHOSTS file description

DBSERVERN. or DBSERVERAI	NETTYPE	Hostname or IP address	ServiceName or port number	Options
HR_prod	onsoctcp	barney	1543	k=0,...

The previous section mentioned that every IDS instance has a name. That name is in the first column of the sqlhosts file. The location of the instance is in the third column, the hostname column. The port number where IDS is listening for incoming SQL requests is in the fourth column.

Column two, the NETTYPE column, allows you to specify whether to use a network protocol or a local protocol for the client to talk to the IDS instance. This is directly related to where the client is running, as opposed to where the IDS instance is running.

If the client and the server are running on the same physical machine, the communication can take place in one of several ways:

- Shared memory (known as inter-process communication or ipc)
- Network interfaces (sockets or TLI)
- Local pipes (named or un-named)
- DRDA (Distributed Relational Database Architecture)

However, if the client and the server are running on different machines, then the communication routes are narrowed down because you have to rely on the network. So, in this case, you only have the options of:

- Network interfaces (sockets or TLI)
- DRDA

The NETTYPE column is how the administrator specifies which communication route a client should use when connecting to an instance by the name specified in column one.

The fifth column, also know as the Options column, is an optional column that can be used to configure various things. The example above has the value k=0. The "k" stands for keep-alive, and the "0" turns this feature off when connecting to the server using this DBSERVERNAME. The keep-alive feature asks the network service to periodically check the receiving end of the connection to make sure that it still exists. If the receiving end doesn't respond in a timely manner, the network service assumes something happened, terminates the connection, and frees up its resources. By default, the keep-alive feature is turned on and should usually be left on.

Table 4 illustrates another example:

Table 4. SQLHOSTS file example

DBSERVERN/ or DBSERVERAI	NETTYPE	Hostname or IP address	ServiceName or port number	Options
HR_prod	onsoctcp	192.168.12.234	1543	b=8192
Acct_devel	onipcshm	dino	dummyvalue	

Based upon the first line in the above example, if the client's `INFORMIXSERVER` environment variable is set to HR_prod, onsoctcp tells the client to use a sockets implementation of the TCP protocol to talk on the network. The client would send any SQL requests to port 1543 of the machine with IP address 192.168.12.234. The hostname field can use either the IP Address or the machine hostname, as long as the hostname can be resolved to an IP address with the appropriate system calls. The option of b=8192 tells the client to use a buffersize of 8192 bytes when communicating with the server. Default connections use a buffersize of 4096 bytes.

If the client's `INFORMIXSERVER` environment variable is set to Acct_devel, onipcshm tells the client that the server is local and to use a special mechanism known as Shared Memory connections. The special mechanism is defined and maintained by IDS for local connections to be able to connect to the instance using a global piece of Unix Shared Memory. The client will then attach to this piece of shared memory, write its SQL requests into it, and read the result sets out of it. As described above, this is only available when the client application and the IDS instance are running on the same machine.

The ServiceName column can use either a port number or a service name. If a service name is used, that value must be able to be resolved to a valid port number, as specified in the /etc/services file on Unix/Linux/Mac or system32\drivers\etc\services on Windows.

Configuration file

During configuration, the ONCONFIG file usually comes hand in hand with the sqlhosts file because of the fields that need to match. As mentioned before, every instance has a name, and that name is used in the first column of the sqlhosts file. How does an instance know what its name is? That is just one of the many parameters that are set up in the IDS configuration file. The IDS install comes with a file called onconfig.std in the etc subdirectory of the installation path. This file should be used as a template for the actual configuration file of an instance. Copy the onconfig.std file to a file of another name, (for example, onconfig.prod). Although the onconfig part of the filename is not required, it has become standard. Once the file has been named, the `ONCONFIG` environment variable should be set to point to that filename.

Note: The `ONCONFIG` environment variable is *only* the name of the file, not the location, because the configuration file has to exist in $INFORMIXDIR/etc (%INFORMIXDIR%/etc on Windows).

At over 180 configurable parameters and over 1100 lines long with comments, the configuration file can be quite daunting. But don't worry; the massive amounts of comments, plus a little experience, will bring it all into perspective. In fact, one of the nice things to know is that only nine of those 180+ parameters are needed to get an instance of IDS up and running. The rest are for performance, extensibility, and feature support.

So, let's start by taking a look at those nine and then maybe a couple more for fun.

The nine important configuration parameters to getting an instance running include:

- `ROOTNAME`
- `ROOTPATH`
- `ROOTOFFSET`
- `ROOTSIZE`
- `MSGPATH`
- `CONSOLE`
- `DBSERVERNAME`
- `DBSERVERALIASES`
- `SERVERNUM`

The first four parameters have to do with disk space. It is not required that all the disk space the instance will use be configured up front. More disk space can be added when needed; however, a certain amount of disk space does need to exist from the very beginning. This disk space is given a name (`ROOTNAME`), a location (`ROOTPATH`), a starting position (`ROOTOFFSET`), and a size (`ROOTSIZE`). When the instance of IDS using this configuration file is started for the very first time, it will format this defined space and initialize it to a specified look. Because of this, you need to make sure that nothing else is using this same disk space. `ROOTPATH` can point to an existing file or a raw device.

The fifth and sixth parameters have to do with logging messages. Since the IDS instance is meant to be something that runs in the background, it needs a location where it can write out informational, warning, and errors messages. `MSGPATH` defines a path and filename of the file you would like the messages to be written to. `CONSOLE` can be used to send special messages to a console screen if one is used. Because of the duplication of messages to these two places, the standard has become to send the `CONSOLE` to /dev/null.

This tutorial has discussed the seventh and eighth parameters before. Every instance of IDS has a name (`DBSERVERNAME`), and it can have more than one name if needed for different types of connectivity (`DBSERVERALIASES`). Think about it this way: you may have the birth name Jeffrey, but you might have other names you respond to as well (aliases), like Jeff. You could say that people who don't know you use Jeffrey, but close friends call you Jeff. You respond to both the same, but you might respond in a different way, depending on which name you were being called. It's similar with IDS—it can have only one name, but can have many aliases. The name and each of the aliases must be listed in the SQLHOSTS file on a different line. Remember that it is the SQLHOSTS file that tells the client how to connect to the instance, depending on the name or alias (first column in SQLHOSTS file) that is used.

The ninth parameter is a unique integer between zero and 255 for each instance of IDS that is running on the same machine. Without going into too much detail, this number (`SERVERNUM`) is used to help generate a value needed for UNIX Shared Memory. Since it is possible to start up more than one instance of IDS running on the same machine, the value specified by `SERVERNUM` has to be different for each instance to help IDS make sure that it calculates a unique number to give to UNIX.

A few other parameters of interest without too much detail include:

- `PHYSFILE`, which helps in sizing the physical log
- `LOGFILES` and `LOGSIZE`, which help in configuring the logical logs
- `ADMIN_MODE_USERS`, which specifies users who can connect while the instance is in Administrative mode; user informix can always connect
- `DBCREATE_PERMISSION`, which specifies a user who can run the `CREATE DATABASE` SQL statement

- BUFFERPOOL, which configures the size and tuning parameters of a bufferpool

So even though the configuration file is huge, taking it piece by piece will help you start to understand it. Remember, you don't need to know everything in it to get an instance of IDS up and running.

Note: With a few minor exceptions, the onconfig file is only read at startup time; any changes made directly to the onconfig file will not take effect until the instance is stopped and restarted. Some of the onconfig parameters can be changed dynamically with the onmode command (discussed later in this tutorial).

Configuration reflection

Note: Now that you have learned about the three parts of configuration—environment variables, sqlhosts file, and the onconfig file—, let's revisit the importance of how these three parts mesh together so that client connectivity works successfully.

Have at an example snippet from each part:

- Environment variables:

```
INFORMIXSERVER=HR_Prod
```

- SQLHOSTS file:

Table 5. SQLHOSTS file example

DBSERVERN/ or DBSERVERAI	NETTYPE	Hostname or IP address	ServiceName or port number	Options
HR_prod	onsoctcp	barney	1543	
HR_Devel	onipcshm	barney	dummyplaceholder	

- onconfig file:

```
DBSERVERNAME       HR_Prod
DBSERVERALIASES    HR_Devel
```

Notice how all three of them have the same "Name" (HR_Prod) in them. The onconfig file tells the instance what its name is. The sqlhosts file tells you where to find the instance by that name. The client says what instance it wants to talk to by specifying it in the environment variable.

So when the client starts running, it takes the value of the INFORMIXSERVER environment variable, looks it up in the sqlhosts file, finds out where to send its SQL request to, and then sends it. From the other side, the DBA named the instance in the onconfig file, and when the instance was started, it looked itself up in the sqlhosts file to find out what servicename (port number) it should be listening to for incoming requests.

Section 4. Starting it up

I don't feel we can end this tutorial until you understand how to complete the configuration and end up with a running piece of software.

What have you done so far:

- Planned for installation
- Installed the product that you decided you needed according to your planning
- Configured an instance of the product, setting up the environment, the sqlhosts file, and the onconfig file
- Made sure the disk space to which `ROOTPATH` is pointed exists

Now it is time to bring this to completion. This last section talks about starting and stopping an IDS instance.

oninit

Since an IDS instance is just a set of OS resources used by a database, you need a way to allocate those resources and deallocate them when necessary. The tools you have to do this are known as the `oninit` and `onmode` commands. Although valid on Windows as well, since an IDS instance runs as a service, it is better to start and stop the service, or use the starts command to help start the service.

Before talking about starting the software, let's consider one more thing. An IDS instance has several states that it can be in. Table 6 lists some of the different states and what they mean:

Table 6. IDS states

State	Description
Offline	Instance is stopped; software is not running.

Fast-Recovery	Instance is starting up and getting to a consistent state from being stopped.
Quiescent	Instance is started, but no users can connect; no SQL can be run.
Administrative	Instance is started; only Admin users can connect and run sql. Also called *single-user* mode.
On-Line	Normal running state; all users can connect and run SQL.
Shutting-down	Instance is stopping; no users can connect; no sql can run.

The `oninit` command is only valid for starting the instance. Think of the instance like a car. To start a car, you turn the key; if the car is in any state except off, turning the key will make a terrible complaining noise. An IDS instance is the same. The `oninit` command only works if the instance is stopped (offline). If the instance is in any other state, issuing the `oninit` command will only complain. It won't hurt anything, but it won't work either.

The `oninit` command comes with the options listed and described in Table 7:

Table 7. oninit options

Option	Description
`-i`	Initializes disk space. Kind of like formatting a hard drive, it should only be used the very first time the instance is started.
`-y`	Automatically answers "yes" to any questions.
`-j`	Starts the instance in Administrative mode. Also known as *single-user* mode.
`-v`	"Verbose" mode; prints out additional messages to stdout while starting.
`-s`	Starts the instance in quiescent mode.

There are more options than this to use for special features, but let's stop with these for this tutorial.

onmode

The `onmode` command is used to stop the instance as well as a multitude of other things.

The `onmode` command is used to change the state of the instance, to dynamically change some of the parameters in the onconfig file, to add and free memory, to configure the B-tree scanner, to configure HDR and Mach11 features, to force a checkpoint, and much more.

This tutorial only covers two of these topics: changing the state of the instance and dynamically changing onconfig file parameters.

When using the `onmode` command to change the state of the instance, it uses the following options:

Table 8. onmode options

Option	Description
`-m`	Brings the instance from single-user or quiescent state to on-line state.
`-s`	Performs a *graceful* shutdown and brings the instance to quiescent state from single-user or on-line state.
`-j`	Performs an *immediate* shutdown for non-admin users only, and brings the instance to single-user state from quiescent or on-line state.
`-u`	Performs an *immediate* shutdown and brings the instance to quiescent state from single-user or on-line state.
`-k`	Performs an *immediate* shutdown and brings the instance off-line from any other state.
`-y`	Automatically answers "yes" to any questions.

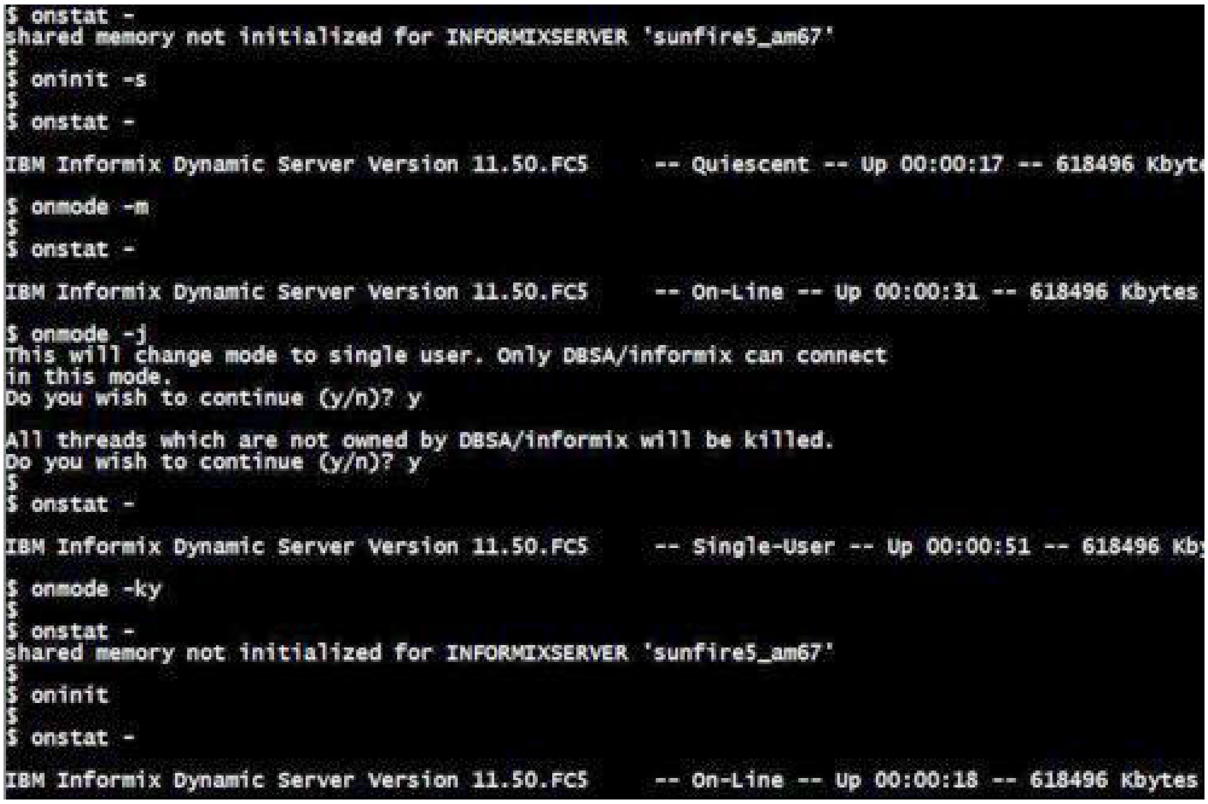
As shown in *italics* in the table above, IDS has two forms of shutting down: a *graceful* shutdown and an *immediate* shutdown.

The graceful shutdown doesn't allow any new connections, but allows currently connected users to continue until they disconnect themselves. When the last user disconnects, the instance will change to the state specified by the option given to the `onmode` command.

The immediate shutdown stops all activity on the database and immediately takes the instance to the state specified by the option given to the `onmode` command.

Figure 1 illustrates running the `oninit` and `onmode` commands. In between each command, an `onstat -` is run to show the state of the instance. The message "Shared memory not initialized for INFORMIXSERVER 'xxx'" means that there is no instance running with that name.

Figure 1. Using the oninit and onmode commands



```
$ onstat -
shared memory not initialized for INFORMIXSERVER 'sunfire5_am67'
$
$ oninit -s
$
$ onstat -

IBM Informix Dynamic Server Version 11.50.FC5     -- Quiescent -- Up 00:00:17 -- 618496 Kbyt

$ onmode -m
$
$ onstat -

IBM Informix Dynamic Server Version 11.50.FC5     -- On-Line -- Up 00:00:31 -- 618496 Kbytes

$ onmode -j
This will change mode to single user. Only DBSA/informix can connect
in this mode.
Do you wish to continue (y/n)? y

All threads which are not owned by DBSA/informix will be killed.
Do you wish to continue (y/n)? y
$
$ onstat -

IBM Informix Dynamic Server Version 11.50.FC5     -- Single-User -- Up 00:00:51 -- 618496 Kb

$ onmode -ky
$
$ onstat -
shared memory not initialized for INFORMIXSERVER 'sunfire5_am67'
$
$ oninit
$
$ onstat -

IBM Informix Dynamic Server Version 11.50.FC5     -- On-Line -- Up 00:00:18 -- 618496 Kbytes
```

onmode -wf/-wm

As mentioned before, the `onmode` command can be used for other things as well, including dynamically changing the values of some of the `onconfig` parameters. This is accomplished with the `onmode –wf` and `onmode –wm` commands. The easiest way to remember the difference is that "f" stands for file and "m" stands for memory. So the `onmode –wf` command changes the current setting in memory and changes the value in the onconfig file. The `onmode –wm` command only changes the current setting in memory.

Note: As of IDS 11.50, only a limited subset of the `onconfig` parameters can be dynamically changed this way.

The syntax is `onmode –wf <onconfig parameter>=<value>` or `onmode –wm <onconfig parameter>=<value>`.

Examples:

```
onmode -wf AUTO_CKPTS=0
```

```
onmode -wm RESIDENT=1
```

If you try to change a value that is not compatible with the `onmode –wf/wm` commands, you will see the following error:
"Configuration Parameter to be changed is not valid or not supported with this option."

Section 5. Summary

This tutorial has covered a plethora of topics, from planning, to installing, configuring, starting, and stopping of the software. Although this may seem overwhelming to some, don't fear. Actually, it is surprising how easy it can be after a little bit of practice.

Just remember that planning out your database installation and setup can help save you loads of revision time later on. It is important to design your database model based on the application type expected—to understand how many users are expected and where they are connecting from, as well as data types and data quantity.

Once a decision has been made on what edition of IDS meets your needs, you can install the product and start configuration of an instance. To get an instance up and running, you don't need to understand every configuration parameter available (you actually only need less than a dozen of them). Make sure that your connectivity is set up correctly between the onconfig file, the sqlhosts file, and the client environment. Always remember to think of everybody as a client; even a DBA is a client to the instance. The DBA might do some slightly different work than an SQL user, but the DBA is still a client to the instance.

After initial configuration is complete, the instance can be started and stopped with the `oninit` and `onmode` commands. The `onmode` command can also help in dynamic configuration of the instance.

Now that you have some of the basics needed to start using IDS, the next tutorial in this series will help you to continue your education on Informix Dynamic Server.

Resources

Learn

- developerWorks Informix zone: Get the resources you need to advance your skills in the Informix arena.
- "The IDS Detective Game" (developerWorks, April 2008): Learn or teach the basics of Informix Dynamic Server (IDS) and relational databases with an interactive game called "The IDS Detective Game".
- IDS roadmap for administrators, developers, and end users: Find resources for all aspects of IDS—planning, installing, configuring, administering, tuning, monitoring, and more.
- Informix Education Training Path: See the courses you need to take to achieve particular skills or certification.
- Informix library: Learn more details about IDS from the online manuals or the IDS Information Center.
- IBM Informix Dynamic Server 11.50 Information Center: Find information that you need to use the IDS family of products and features.
- developerWorks Information Management zone: Learn more about Information Management. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with developerWorks technical events and webcasts.

Get products and technologies

- Informix Dynamic Server Express Edition: Download a trial version of Informix Dynamic Server Express Edition to get started with IDS.
- Informix Dynamic Server Enterprise and Developer Edition: Download a free trial version of Informix Dynamic Server Enterprise or Developer Edition.
- Download IBM product evaluation versions or explore the online trials in the IBM SOA Sandbox, and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Participate in the discussion forum for this content.
- IDS Experts Blog: Read the technical notes on Informix Dynamic Server from a worldwide team of Development and Technical Support engineers.
- Participate in developerWorks blogs and get involved in the My developerWorks community; with your personal profile and custom home page, you can tailor

developerWorks to your interests and interact with other developerWorks users.

About the author

Jeffrey S. Bohm
Jeff has been working with Relational Database Management Systems (RDBMS) products for over 16 years. He has experienced databases from every angle: as a customer, a consultant, an instructor, and a developer. He was hired on at Informix Software in 1994 and has been working with it ever since. His jobs at Informix included advanced technical support, performance tuning and troubleshooting consultant, and customer education specialist. He currently holds a Software Engineer position with the IBM Informix Dynamic Server Stress QA team.

Jeff has worked with hundreds of IDS clients, from the smallest of businesses to the largest, including multiple large retail, grocery, and telco companies. He has helped them get the most out of their IDS installs, addressing the most basic SQL needs to the most demanding performance and feature rich implementations.

Jeff was involved in writing the IDS 9 Certification exams and currently holds Certifications for IDS 5, 7, 9, 10, and 11.

Jeff also helped author several classes that are being taught in IBM's IDS and RDBMS curriculum.