Informatica

Informatica® Data Clustering Engine 10.5

User Guide

Informatica Data Clustering Engine User Guide
10.5
September 2022

Publication Date: 2023-09-28

Table of Contents

Preface 7

Informatica Resources 7

- Informatica Network 7
- Informatica Knowledge Base 7
- Informatica Documentation 7
- Informatica Product Availability Matrices 8
- Informatica Velocity 8
- Informatica Marketplace 8
- Informatica Global Customer Support 8

Chapter 1: Introduction 9

Clustering Process 9

Data Clustering Engine 9

Terminology 10

Clustering Suite 11

Chapter 2: Installation 13

Software Prerequisites 13

- Verify the Minimum System Requirements 13
- Install Java 13
- Set the Environment Variables 14
- Install the .NET Framework 14
- Install the ODBC Drivers 14
- Extract the Installer Files on Windows 14
- Extract the Installer Files on UNIX 14
- Set Up the X Window Server on UNIX 15

Software Installation 15

Post-Installation Tasks 15

- Configuring an ODBC Data Source 15
- Testing the Database Configuration 18
- Testing the Installation 19
- Cloning a Sample Project from a definition file (.SDF) 19
- Cloning a Sample Project from a template file (.pr) 19

Chapter 3: Design 21

Syntax 21

Project Definition File 22

SYSTEM Section 22

- IDT Definition 23
- IDX Definition 24

Logical File Definition. . . . 25
Clustering Definition. . . . 27
Search Definition. . . . 29
Multi-Search Definition. . . . 31
Multi-Clustering-Definition. . . . 32
User-Job-Definition. . . . 33
User-Step-Definition. . . . 34
The Job Definition. . . . 35
Search Logic. . . . 41
Score Logic. . . . 44
FILES and VIEWS Sections. . . . 46
Data Source. . . . 46
File Definition. . . . 46
View Definition. . . . 46
Syntax. . . . 47
Field Formats. . . . 47
Transformations. . . . 49
Cluster File. . . . 53
Output Views. . . . 53
USER SOURCE TABLE Section. . . . 54
General Syntax. . . . 54
UST Data Types. . . . 55
Single UST. . . . 57
Joining USTs. . . . 61
Merging USTs. . . . 63
Defining Source Tables. . . . 64
Flattening IDTs. . . . 65
Syntax. . . . 68
Flattening Process. . . . 69
Flattening Options. . . . 69
Tuning / Load Statistics. . . . 69
Maintaining the Project. . . . 70
Restarting or Continuing an Edit Session. . . . 70
Editing a Project. . . . 71
Project Editor. . . . 71
Starting. . . . 71
Editing. . . . 72
Cloning and Adding. . . . 72
Help. . . . 72
Advanced Options. . . . 72
Project Template. . . . 72
Backing Up the Database and Index. . . . 73

Membership Attributes. . . . 73
Performance Optimization. . . . 74
Partitions. . . . 75
Key Data / Key-Score-Logic. . . . 75
Pre-Scoring. . . . 76
Scoring Phases. . . . 76
Utilizing multiple CPUs. . . . 77
Reducing Database I/O. . . . 78
Large File Support. . . . 84

Chapter 4: Operation. . . . 86

Environment Variables. . . . 86
Limitations. . . . 89
DCE Console. . . . 90
Running the Console Server. . . . 90
Running the Console Client. . . . 90
Settings. . . . 91
The Main Console Window. . . . 93
The Options Explained. . . . 93
How to Create and Run Jobs. . . . 98
Job Options. . . . 98
How to Run Clustering. . . . 101
Requirements Analysis. . . . 101
Launch the Console Server and DCE Console. . . . 101
Create Project Definition. . . . 102
Confirm Settings. . . . 102
Create the Project. . . . 102
Select (and Load) the Project. . . . 104
Run the Clustering Process. . . . 104
Review Results. . . . 106
Adjust parameters and rerun if required. . . . 106
Stopping and Restarting Clustering. . . . 106
Stopping and Restarting Clustering Manually. . . . 107
To restart a suspended job. . . . 109
Stopping and Restarting Clustering Automatically. . . . 110
Utilities. . . . 110
Batch Search Client - Relate. . . . 110
Report Formats. . . . 112
Delimited Input. . . . 113
SQL Input. . . . 114
Batch Search Client - DupFinder. . . . 114
Extra options for Relate and DupFinder. . . . 115
Threads. . . . 116

Batch Process - Command Line. 116
Report Viewer. 117
Troubleshooting. 118
How To. 122

Index. 128

Preface

The Data Clustering Engine Guide provides information about the DCE product.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at http://velocity.informatica.com. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

CHAPTER 1

Introduction

This chapter discusses the general concepts and terminology associated with the Data Clustering Engine (DCE).

Clustering Process

Clustering is the process of grouping similar or related records together. The logical rules, which define how to cluster records together, are specified using definition files and are stored in a special database called Rulebase.

For example, data records could be processed into the following groups:

- all records about the same person
- all records about the same family
- all records about the same household
- all records which have the same demographic attributes

Data may be clustered in many ways. Simple forms of clustering include records which:

- have the same account number
- have an identical name
- have the same match-code

Complex forms of clustering may include records which:

- could be about the same house
- could be safely accepted as the same person
- could possibly refer to the same person

Data Clustering Engine

The DCE is a suite of software application programs that facilitate the clustering process using userdefined rules together with SSA-NAME3 search and matching technology.

The DCE reads input files and produces a database containing clustered records. The clusters can be output to reports and/or read directly via an Application Programming Interface.

The DCE consists of:

- an input file processor
- a proprietary Rulebase and Database
- sort and merge utilities
- clustering programs
- report and statistics generator

The core of the DCE consists of a Rulebase and a Database. During the clustering process, the Rulebase is loaded with information pertaining to how the data should be matched combined with the SSANAME3 algorithms to be used for , Searching and Matching. The Database is loaded with records from input file(s).

The Input File Processor and sort-merge utilities can be used to help load the database and optimize its efficiency. The Input File Processor can reformat and restructure the input data for better clustering. The Sort-Merge utilities can be used to sort the input file by key value. Loading the database in this sequence helps to optimize I/O performance while clustering the data.

The Report and Statistics Generator helps to produce reports from the clustered data which can be viewed from the DCE Console using the integrated report viewer tool.

Terminology

The DCE clusters records based upon user-defined rules. These rules are read from definition files and loaded into the Rulebase. The following entities will be defined when creating a definition file.

Project

A Project represents the complete logical task that one wants to accomplish. It is used to record information that affects the entire process. Many Projects may be defined per Rulebase and each Project may contain many clusterings.

IDT

An Identity Table (IDT) is a DCE table stored in the database containing denormalized and transformed data. It is created by extracting data from either database tables (User Source Tables) or a sequential (flat) file.

IDX

An Identity Index (IDX) is an index stored in the database containing keys for each row of the IDT. It may optionally store additional matching data (Key-Data) together with each key in order to improve clustering performance.

Clustering

A clustering defines the application objectives and business rules for a single clustering process. It will define the search and matching rules to be used.

The clustering also defines which DCE utilities are used to create the clustering result. This is done by associating jobs with a clustering.

Job

A job represents a unit of work performed by the DCE. For example it might be the process of converting the input file, loading it into the database, or clustering the records on the database.

A job entity is used to specify information that is particular to a job. If the job accesses an input or output file, the job will refer to a logical-file entity.

Logical File

A logical-file entity is used to specify how the DCE will access a physical file. It describes the path to the file, the file format, and the view to be used when accessing the file.

View

A view specifies the layout of a record and the rules for constructing and using that record. A view definition controls what information is loaded to - or extracted from - the database.

Clustering Suite

The clustering suite consists of the following programs:

PRE

This program performs the pre-clustering formatting step. It is used to read the raw input file using a view and creates records that conform to the database view of the data. This process can optionally add a unique Source-Id field to each record if the input data does not already contain one. `SORTIT`, `LOADIT` and `CLUSTER` are also capable of reading the input files, so under normal circumstances the `PRE` step would be omitted. However, `PRE` should be run when you wish to process multiple input files, each with a different format and/or you wish to allocate a different Source-Id to each input file.

SORTIT

This program can be used to sort the input file. This step is optional and is used to improve the performance of the clustering phase. The input file can be either the output from the `PRE` program, or the raw input-file. If the latter is the case, `SORTIT` will perform the `PRE` process (internally) prior to sorting the records.

The records are sorted using the key field specified in the Clustering Definition.

Note: Sorting is used to reorder the input file by preferred key in order to place similar records close together on disk, thereby improving performance due to locality of reference. However, if you use negative keys or a negative search strategy, which is the default strategy in SSA-NAME3 standard populations, you'll be searching randomly over the disk, thereby negating the benefit. A `SORTIT` step should be omitted in this situation.

LOADIT

This program pre-loads the input data to the database and generates keys (which are stored in a database index). It is an optional process. It can read its input from either

a raw input file

the output from `PRE`

the output from `SORTIT`

If `LOADIT` reads its input from a raw input file it will internally perform the `PRE` process prior to loading the records to the database.

CLUSTER

This program clusters the data records using the rules specified in the Clustering Definition. Data records can be read from:

a raw input file

the output from PRE

the output from SORTIT

the database (output from LOADIT)

The result of the clustering process is held in the database in the form of the cluster relationship index.

EXTRACT

This program is used to create a database index required by the POST program. Normally this index is created during the CLUSTER phase, unless you specify the DELAY option. In this case, you must schedule an EXTRACT job prior to running POST.

POST

This is a general-purpose post-clustering extraction and reporting step. The layout of the report/file is controlled using a sophisticated view processor

CHAPTER 2

Installation

This chapter includes the following topics:

- Software Prerequisites, 13
- Software Installation, 15
- Post-Installation Tasks, 15

Software Prerequisites

Before you install Data Clustering Engine, set up the machine to meet the requirements to install and run Data Clustering Engine. If the machine does not meet the pre-installation requirements, the installation might fail.

Verify the Minimum System Requirements

Verify that the machine meets the hardware and software requirements to install Data Clustering Engine.

The following table describes the minimum system requirements to install and run Data Clustering Engine:

Component	Minimum requirement
CPU Cores	4
RAM	8 GB

The hardware requirements for a Data Clustering Engine implementation depend on the search strategies and the number of identity tables, identity indexes, and concurrent users that you plan to use.

For more information about product requirements and supported platforms, see the Product Availability Matrix.

Install Java

Install Java Development Kit (JDK) or Java Runtime Environment (JRE) version 1.6 or later.

Set the Environment Variables

The following table describes the environment variables that you must configure if the variables are not already configured:

Variable	Description
JRE_HOME	Specify the location of the directory that contains the supported JRE.
PATH	Include the location of the `bin` directory of the JDK or JRE installation.
JAVA_OPTS	Optional. Additional parameters that you want to pass to the JRE.

Install the .NET Framework

On Windows, install the .NET Framework version 2 or later.

Install the ODBC Drivers

If you use a database to source data, Data Clustering Engine requires an ODBC API to access the database.

Unless you use Oracle, install and configure the ODBC drivers for your database. Data Clustering Engine does not use the Oracle ODBC driver and uses custom ODBC drivers for Oracle, `ssaoci9.dll` on Windows and `libssaoci9.so` on UNIX. The custom drivers do not require any further configuration.

Extract the Installer Files on Windows

On Windows, the installer files are compressed and distributed as a ZIP file.

Use a ZIP utility to extract the installer files to a directory on your machine.

You can extract the installer files using FTP download. Download the Informatica installation ZIP file from the Informatica Electronic Software Download (ESD) site to a directory on your machine and then extract the installer files.

When you download the installation files from the Informatica Electronic Software Download (ESD) site, the license key is in an email message from Informatica. Copy the license key file to a directory accessible to the user account that installs the product.

Contact Informatica Global Customer Support if you do not have a license key or if you have an incremental license key and you want to create a domain.

Extract the Installer Files on UNIX

On UNIX, the installer files are distributed as a TAR file.

Use a native tar or GNU TAR utility to extract the installer files to a directory on your machine.

You can extract the installer files using FTP download. Download the Informatica installation TAR file from the Informatica Electronic Software Download (ESD) site to a directory on your machine and then extract the installer files.

When you download the installation files from the Informatica Electronic Software Download (ESD) site, the license key is in an email message from Informatica. Copy the license key file to a directory accessible to the user account that installs the product.

Contact Informatica Global Customer Support if you do not have a license key or if you have an incremental license key and you want to create a domain.

Set Up the X Window Server on UNIX

On UNIX, when you run the installer in graphical mode, use a graphics display server. The graphics display server is typically an X Window server. If you do not have the X Window server installed on the machine where you want to install the product, you can use an X Window server installed on another machine or run the installer in the console mode.

Software Installation

Follow the instructions in the *Informatica Identity Resolution Installation and Configuration Guide* to install Data Clustering Engine.

Post-Installation Tasks

After you install Data Clustering Engine, perform the following tasks:

1. If you use a database to source data, configure the OBDC data source for your database, and test the database configuration.
2. Test the Data Clustering Engine installation.
3. Clone a sample project from a system definition file or a template file.

Configuring an ODBC Data Source

If you use a database to source data, Data Clustering Engine requires an ODBC API to access the database. During the run time, the database access layer of Database Clustering Engine tries to load an appropriate ODBC driver for each database type. You must create ODBC data sources for each database that you plan to access.

Oracle

Data Clustering Engine uses the following custom ODBC drivers for Oracle instead of the Oracle ODBC driver.

- On Windows: `ssaoci9.dll`
- On UNIX: `libssaoci9.so`

When you install Data Clustering Engine, the installer updates the `odbc.ini` file in the `<Data Clustering Engine Installation Directory>\bin` directory on Windows or `<Data Clustering Engine Installation Directory>/bin` directory on UNIX. The `odbc.ini` file contains the following entries, and you can edit them based on your requirement:

```
[<Service Name>]
driver = <ODBC Driver>
server = <Native Database Service>
```

Configure the following parameters:

Service Name

Service name that Data Clustering Engine uses to refer to the database service.

ODBC Driver

Name of the ODBC driver, which is `ssaoci9.dll` on Windows and `libssaoci9.so` on UNIX. The driver name does not include the directory path.

Note: In Data Clustering Engine versions earlier than 9.5.2, you can also use the `ssadriver` parameter on Windows or the `ssaunixdriver` parameter on UNIX instead of the `driver` parameter. The `ssadriver` and `ssaunixdriver` parameters are deprecated, and Informatica recommends that you use the `driver` parameter.

Native Database Service

Name of the database service that you configure in the database.

You can find a sample configuration file, `odbc.ini.ori`, in the following directory:

- On Windows: `<Data Clustering Engine Installation Directory>\bin`
- On UNIX: `<Data Clustering Engine Installation Directory>/bin`

When you use `ssaoci9.dll` driver with Oracle Database Client 10g, the connectivity test might fail on UNIX environments. The `ssaoci9.dll` driver is linked with `libclntsh.so.9.0`, which is not distributed with Oracle 10g. This issue causes the connectivity test failure. Oracle adds symbolic links to redirect requests for older versions of a library to the current version. Oracle provides this backward compatibility only to the minor versions of its release. To overcome this problem, identity the appropriate Oracle library directory such as `lib`, `lib32`, or `lib64`, and add the symbolic link as follows:

```
cd $ORACLE_HOME/lib32
ln -s ./libclntsh.so libclntsh.so.9.0
```

Microsoft SQL Server

1. Open the `odbc.ini` file that you can find in the `<Data Clustering Engine Installation Directory>\bin` directory, and add or update the following entries:

   ```
   [<Service Name>]
   DataSourceName = <ODBC DSN>
   driver = <ODBC Driver>
   server = <Native Database Service>
   ```

   Configure the following parameters:

   Service Name

   Service name that Data Clustering Engine uses to refer to the database service.

   ODBC DSN

   Data Source Name. If you do not specify any value, it defaults to the service name.

   ODBC Driver

   Name of the ODBC driver that communicates with the database. The driver name does not include the directory path. You can also use the Microsoft SQL Server Native Client as the ODBC driver.

   Enter one of the following values:

   - `sqlsrv32` for any supported version of Microsoft SQL Server
   - `sqlncli` for Microsoft SQL Server 2005
   - `sqlncli10` for Microsoft SQL Server 2008

- `sqlncli11` for Microsoft SQL Server 2012

Note: In Data Clustering Engine versions earlier than 9.5.2, you can also use the `ssadriver` parameter instead of the `driver` parameter. The `ssadriver` parameter is deprecated, and Informatica recommends that you use the `driver` parameter.

Native Database Service

Name of the database service that you configure in the database.

Note: You can find a sample configuration file, `odbc.ini.ori`, in the `<Data Clustering Engine Installation Directory>\bin` directory.

2. Open the ODBC Data Source Administrator by performing the following tasks:
   a. On the **Start** menu, click **Control Panel**.
   b. In **Control Panel**, click **Administrative Tools**.
   c. In **Administrative Tools**, click **Data Sources (ODBC)**.
3. On the **User DNS** tab or **Service DNS** tab, click **Add**.
4. Select **SQL Server** or **SQL Server Native Client**, and click **Finish**.
5. Follow the instructions in the Wizard, and ensure that you enter the following values:
   - Specify a data source name. Ensure that the data source name matches the data source name that you specify in the `odbc.ini` file.
   - Select the Microsoft SQL Server authentication and provide the user credentials that you created.
   - Specify a default database.
   - Ensure that you do not select the **Use regional settings when outputting currency, numbers, dates and times** check box.

   Note: If you want to use the Windows authentication to connect to Microsoft SQL Server, set the SSA_DB_WINDOWS_AUTHENTICATION environment variable to Yes.

IBM DB2 UDB

1. Open the `odbc.ini` file that you can find in the `<Data Clustering Engine Installation Directory>\bin` directory on Windows or `<Data Clustering Engine Installation Directory>/bin` on UNIX, and add or update the following entries:

```
[<Service Name>]
DataSourceName = <ODBC DSN>
driver = <ODBC Driver>
server = <Native Database Service>
```

Configure the following parameters:

Service Name

Service name that Data Clustering Engine uses to refer to the database service.

ODBC DSN

Data Source Name. If you do not specify any value, it defaults to the service name.

ODBC Driver

Name of the ODBC driver. The driver name does not include the directory path. Enter one of the following values:

- `db2cli` on Windows

- db2 on UNIX
- libdb2.a(shr.o) on 32-bit AIX environments
- libdb2.a(shr_64.o) on 64-bit AIX environments

Note: In Data Clustering Engine versions earlier than 9.5.2, you can also use the ssadriver parameter on Windows or ssaunixdriver parameter on UNIX instead of the driver parameter. The ssadriver and ssaunixdriver parameters are deprecated, and Informatica recommends that you use the driver parameter.

Native Database Service

Name of the database service that you configure in the database.

Note: You can find a sample configuration file, odbc.ini.ori, in the following directory:

- On Windows: <Data Clustering Engine Installation Directory>\bin
- On UNIX: <Data Clustering Engine Installation Directory>/bin

2. Open the ODBC Data Source Administrator by performing the following tasks:
   a. On the **Start** menu, click **Control Panel**.
   b. In **Control Panel**, click **Administrative Tools**.
   c. In **Administrative Tools**, click **Data Sources (ODBC)**.
3. On the **User DNS** tab or **Service DNS** tab, click **Add**.
4. Select **IBM DB2 ODBC DRIVER**, and click **Finish**.
5. Follow the instructions in the Wizard, and ensure that you enter the following values:
   - Specify a data source name. Ensure that the data source name matches the data source name that you specify in the odbc.ini file.
   - Specify a default database.

Testing the Database Configuration

If you use a database to source data, verify the configuration to ensure that Data Clustering Engine communicates with the database.

You can use the dblist utility to verify your ODBC configuration. You can find the dblist utility in the following directory:

- On Windows: <Data Clustering Engine Installation Directory>\bin
- On UNIX: <Data Clustering Engine Installation Directory>/bin

The dblist utility uses the following syntax:

```
dblist -c -dodb:99:<User Name>/<Password>@<Service Name>
```

This utility uses the following parameters:

User Name

Name of the user that you created.

Password

Password for the user.

Service Name

Service name that Data Clustering Engine uses to refer to the database service.

The following text shows a sample response of the dblist utility:

```
dblist -c -dodb:99:ssa/SSA@ora920
Maximum connections per module: 1024
Linked databases: odb: sdb:
Driver Manager: 'Informatica ODBC Driver Manager <revision>'
ODBC Driver: 'ssaoci9 SSADB8 2.7.0.00GCCLIBC2Feb 1 2010 14:22:57'
DBMS Name: 'Oracle DBMS (9.2.0.6.0)'
Native DB type: 'ora'
*** Connection successful ***
```

Testing the Installation

Launch the DCE Server in configuration mode from the shortcut in the Informatica's folder created during the software installation phase. Launch the **Console Client** in Admin mode. A quick test will start automatically to verify the integrity of the installation.

Refer to the *Using the Console Client* section in Setup Mode for more detailed description of the installation tests.

Cloning a Sample Project from a definition file (.SDF)

An easy way to get started on a new Project is to clone a sample Clustering Project that is similar to your needs. The following steps are necessary to clone a Project from the `dce/tests` directory:

1. Create a new Project work directory (if you do not wish to use the default work directory `dce`).
2. Set the environment variable *SSAWORKDIR* to refer to the work directory

   This can be done by modifying the DCE Server environment setting script `<DCE Installation Directory>\env\dces.bat` on Windows and `<DCE Installation Directory>/env/dces` on Unix-based systems.
3. Copy a sample Project file file `dce/tests/testnn.sdf` to the work directory
4. Start the DCE Server and Console
5. Select **Project** > **New** > **Create a Project** from an SDF and enter the name of your Project and the location of your Project definition file.
6. After the new Project has been created, select it to make it the active Project (**Project** > **Select**)

   Refer to the *DCE Console* section for details.

Cloning a Sample Project from a template file (.pr)

An alternative to using a Project definition file (SDF) is to create a Project by importing a Project template and to modify it using the Project editor included in the Console. The following steps are necessary to create a Project from a previously created template file:

1. Create a new Project work directory (if you do not wish to use the default work directory `dce`).
2. Set the environment variable *SSAWORKDIR* to refer to the work directory.

   This can be done by modifying the DCE Server environment setting script `<DCE Installation Directory>\bin\dces.bat` on Windows and `<DCE Installation Directory>/env/dces` on Unix-based systems.
3. Start the DCE Server and Console.
4. Select **Project** > **New** > **Import a Project**from a flat file and enter the name of your Project and the location of the previously created (exported) Project template file.
5. After the new Project has been created, select it to make it the active Project (**Project** > **Select**).

6. To start the Project Editor, click the **Project** > **Edit** button. Refer to the *Editing a Project* section for more details.

   If the clone Project has a different name than the original Project, it must be loaded either from Console Client (**Project** > **Load**) or from the Project Editor (`Load`), before it can be used with the new name. The load is a necessary step to create the clustering file for the new Project.

CHAPTER 3

Design

This chapter discusses the design of a DCE Project and includes information on creating and modifying the Project Definition file (`.sdf`).

Syntax

A text based Project Definition File (`.sdf`) is used to specify options and parameters which define your Project, clusterings, jobs, logical files, database files and views.

The Project Definition File has the following syntax:

- Each line is terminated by a newline.
- Each line has a maximum length of 255 bytes.
- Lines starting with an asterisk are treated as comments.
- All characters following two asterisks (**) on a line are treated as comments.

Quoted strings can be continued onto the next line by coding two adjacent string tokens. For example, the COMMENT= keyword is another way of defining a comment inside the definition files. If a COMMENT field is very long, it could be specified like this:

```
COMMENT="This is a very long comment that"
" continues on the next line."
```

Some definitions will require the specification of a character such as a filler character. It will be denoted as `<char>` in the following text. The Definition Language permits a `<char>` to be specified in a number of ways:

- `c` - a printable character
- `"c"` - the character embedded in double quotes (")
- `numeric(dd)` - the decimal value dd of the character in the collating sequence
- `numeric(x'hh')` - the hexadecimal value (hh) of the character in the collating sequence

Numeric data in the `Project Definition File` may be suffixed by an optional modifier to help specify large numbers:

- k - units of 1000 (one thousand)
- m - units of 1000000 (one million)
- g - units of 1000000000 (one billion)
- k2 - units of 1024 (base 2 numbers)
- m2 - units of 1048576

- g2 - units of 1073741824

The Project Definition File is created using a text editor. Although it is recommended that you begin with one of the example Project definition files supplied in the `dce/tests` subdirectory on the DCE Client. The Project Definition File is then loaded into the DCE Rulebase during the **Project** > **New step**.

After the initial load, the Project Definition File is maintained using the Project Editor tool in the DCE Console. Refer to the *Editing a Project* section for more details.

Project Definition File

This file contains 4 sections, each identified by a section heading:

```
Section: System Section: Files Section: User-Source-Tables Section: Views
```

A Project must have one Section System, at least one Section Files or User-Source-Table (or both) and one optional Section Views.

SYSTEM Section

This section provides information about the System section.

Project Definition

This section begins with the Project-Definition keyword. The fields are as follows:

Field	Description
NAME=	A character string that defines the name of the Project. This is a mandatory parameter.
ID=	This is one or two digit number to be assigned to the Project. It must be unique (different from other Projects defined to the same Rulebase). Rulebase or Projects in the same work directory. The ID is used to name the clustering relationship indexes. If they are not unique, the clustering of one Project will overwrite the clustering relationship indexes of another Project with the same ID. This is a mandatory parameter.
FILE=	A character string that specifies the name of the DCE Identity Table. If this parameter is omitted, then you must specify an otherwise optional `idt-definition`. For more information, see the *IDT Definition* section below.
OPTIONS=	An optional parameter used to set global options for all ID-Tables in this Project. Currently, the only valid value is `Compress-Method`, which is described in the *IDX Definition* section.
COMMENT=	This is an optional text field that is used to describe the Project's purpose.

Field	Description
DEFAULT-PATH=	DCE will create various files while processing some jobs. The PATH parameters can be used to specify where those files will be placed. DCE will use the path specified in the DEFAULT-PATH parameter. If DEFAULT-PATH has not been specified, the current directory will be used. It is valid to specify a value of "+" for the path. This represents the value associated with the environment variable *SSAWORKDIR*. This is especially recommended when running the Project remotely, ie. from a directory other than the *SSAWORKDIR* directory. **Note:** DCE does not support spaces in file or PATH names.
FORMATTED-FILE-PATH=	Optional path for the formatted data file `fmt.tmp`. Refer to the *Reformat Input Data* section.
INDEXES-PATH=	The path for all index files except the Rulebase index (default name `rule.ndx`), which is always kept in the same directory as the Rulebase (default name `rule.db`).
CLUSTER-IDX1-PATH=	The path to the index file created by the `CLUSTER` job.
CLUSTER-IDX2-PATH=	The path to the index file created by the `CLUSTER/EXTRACT` job.
SORTED-FILE-PATH=	Optional path for the temporary sort data file `srt.tmp`.
SORT-WORK1-PATH=, SORT-WORK2-PATH=	DCE may create sort work files when sorting a large result set. These optional parameters control the placement of these files and override the directory given in `DEFAULT-PATH`.

IDT Definition

This optional section begins with the IDT-DEFINITION keyword and defines a DCE Identity Table. If you don't need to specify any other IDT parameters except its name, then you can omit this section and specify the IDT name using the FILE= keyword under the *Project-Definition* section (see the *Project Definition* section above).

The fields are as follows:

Field	Description
NAME=	A character string that specifies the name of the IDT. This is a mandatory parameter (unless the whole *IDT-DEFINITION* section is omitted).
DENORMALIZED-VIEW=	A character string that specifies the name of the `View-Definition` that defines the layout of the denormalized data for Flattening. Refer to the *Flattening IDTs* section for details. This is an optional parameter.
FLATTEN-KEYS=	A character string that defines the columns used to group denormalized rows during the Flattening process. Refer to the *Flattening IDTs* section for details. This is an optional parameter.
OPTIONS=	This is used to specify various options to control the data stored in the IDT: `FLAT-KEEP-BLANKS` - a flattening option that maintains positionality of columns in the flattened record. Refer to the *Flattening IDTs* section for details. `FLAT-REMOVE-IDENTICAL` - a flattening option that removes identical values from a flattened row. Refer to the *Flattening IDTs* section for details.

IDX Definition

This begins with the IDX-Definition keyword and defines a DCE Identity Index (IDX).

The IDX Definition fields are as follows:

Field	Description
NAME=	A character string that specifies the name of the IDX. This is a mandatory parameter.
COMMENT=	An optional free-form description of this IDX.
ID=	A two-letter character string used to generate the names of the actual database table that represents the IDX. Each IDX must have a unique ID. This is a mandatory parameter.
KEY-INDEX	Optional parameter to specify the name of the index file. If omitted, the value of the `ID=` field will be used.
KEY-INDEX-PATH=	Optional parameter to specify the location of the index file. If omitted, the index will be created in the default work directory. **Note:** DCE does not support spaces in file or PATH names
IDT-NAME=	This is the name of the IDT (as defined in the `File-Definition` or `User-Source-Table` sections) that this IDX will index. This is a mandatory parameter.
AUTO-ID-FIELD=	This field is not required if loading data from a User Source Table. This is the name of a field defined in the *Files* Section that contains a unique record label referred to as the Record Source If no such field exists in the IDT, DCE can generate one. If DCE is being asked to generate an Id, the user can choose the name of the `AUTO-ID-FIELD`, however that name must be defined as a field in the Files section (if using a transform clause, this happens automatically). **Note:** Refer to the *AUTO-ID-NAME=* section under *Logical File Definition*.
KEY-LOGIC=	This is a mandatory parameter describing the key-logic to be used to generate keys for the IDT. It may differ from the `SEARCH-LOGIC=` defined in the `Search-Definition` or `Clustering-Definition`. For more details refer to the *Search Logic* section.
PARTITION=(field[,length,offset[,null-field-value]]),. . .	For very large files, the key may not be selective enough (retrieves too many records). This option instructs DCE to build a concatenated key from the Key-Field (defined by `KEY-LOGIC=`) and up to five fields/sub-fields taken from the record. The `field`, `length` and `offset` represent the field name, number of bytes to concatenate to the key and offset into the field (starting from 0) respectively. The `length` and `offset` are optional. If omitted, the entire field is used. `null-field-value` is used to specify the value which represents an empty field. It defaults to spaces. Records that contain a null partition value can be ignored by specifying the NO-NULL-PARTITION option (defined below).

Field	Description
KEY-DATA=(field[,length,offset]),. . .	Key-data is used to append redundant information from the data record to the key generated from the key-field. Specifying key-data enables the search logic routine to access and test that data, and therefore be able to quickly reject some records from the search candidate list rather than pass them to the more expensive score logic routine. It is a performance option. Key-data uses the same syntax as the PARTITION= parameter to specify up to five fields, or parts thereof, to be appended to the key. The Key-Score-Logic module can examine this data and return a result without the need to read the IDT record. If key-scoring rejects records at this early stage, the scoring overhead is reduced and the clustering's performance is improved. **Note:** See also *FULL-KEY-DATA* option below.
OPTIONS=	This is used to specify various options to control the keys and data stored in the IDX. - `ALT` - Generate and store multiple keys for each record. This is the default value. When disabled only the first key returned by SSA-NAME3 is stored in the IDX. - `FULL-KEY-DATA` - Store all IDT fields in the IDX. The data is stored in uncompressed form unless you specify the Compress-Key-Data option. This option is an alternative to specifying *KEY-DATA* (see above). - `COMPRESS-KEY-DATA(n)` - Store all IDT fields in compressed form using a fixed record length of `n` bytes. `n` cannot exceed (256 - PartitionLength - KeyLength - 4). **Note:** `COMPRESS-KEY-DATA` implies `FULL-KEY-DATA`. - `COMPRESS-METHOD(n)` Methods are: `Method 0` (the default) will store overflow data as multiple adjacent index records. This improves performance as it takes advantage of the locality of reference in the DBMS' cache. `Method 1` will truncate compressed Key Data if its size exceeds the limit defined by the COMPRESS-KEY-DATA option. This will cause additional I/O to access the data record, when necessary. - `NO-NULL-FIELD` do not store records that contain a Null-Field, as defined in the Key-Logic section. - `NO-NULL-KEY` do not store Null-Keys. A Null-Key is defined in the Key-Logic section. - `NO-NULL-PARTITION` do not store keys that contain a Null-Partition value, as defined by the PARTITION= keyword.

Logical File Definition

This begins with the Logical-File-Definition keyword. The fields are as follows:

Field	Description
NAME=	A character string that identifies the Logical-File. A Job object refers to a logical-file with its FILE= parameter. This is a mandatory parameter.
COMMENT=	A character string for documentation purposes only.
PHYSICAL-FILE=	A character string that specifies the file name containing the input data. When reading input from a flat file, this will specify the full filename including the path. When reading input from User Source Tables it specifies the name of the IDT defined in the `CREATE_IDT` clause of the *User Source Table* section. The name should be enclosed in double-quotes. This is a mandatory parameter. **Note:** DCE does not support spaces in file or PATH names.

Field	Description
VIEW=View	The name of the Database View Definition to be used when reading the flat input file. It must not be specified if data will be read from User Source Tables. The View is used to translate the contents of the input file into the layout to be stored in the IDT (specified by the KEY-INDEX= parameter of the `IDX-Definition`).

[INPUT-]FORMAT={SQL|Text|Binary|Delimited}	Describes the format of the input file. When reading from a User Source Table, specify a format of `SQL`. Otherwise, when reading from a flat-file, the following options may be used:
	- `Text` files contain records terminated by a newline.
	- `Binary` files do not contain line terminators. Records must be fixed in length and match the size of the input View.
	- `Delimited` files contain variable length records and fields which are delimited. By default, records are separated by a newline, fields are separated by a comma (,) and the field delimiter is a double-quote ("). However, you may change this behaviour by defining
	`FORMAT=Delimited, Record-Separator(<char>), Field-Separator(<char>), Field-Delimiter(<char>)`
	All three values are always used when the Delimited format is processed. There is no way of specifying that a particular delimiter should not be used. However, you may specify a value that does not appear in your data such as a non-ASCII value. For example, if a field delimiter is not used, the following could be specified:
	`Field-Delimiter(Numeric(255))`

Record-Separator=(<char>)Field-Separator=(<char>)	These parameters are usually specified as sub-fields on the `FORMAT` definition. However, for convenience, they may be defined as separate fields. **Note:** For a delimited file using the default delimiters, there is no need to specify any of the delimiters in either the Project (`*.sdf`) file or in the Relate dialog. The delimiter definition should only be used when re-defining the default delimiters.
AUTO-ID-NAME=	This parameter is used when DCE has been requested to generate a Record Source ID field by defining the AUTO-ID clustering option (see the *OPTIONS=* section for description of this option). The generated ID is composed of a text string concatenated to a base 36 sequence number. The value for the text string portion is specified by using the AUTO-ID-NAME= parameter. It is limited to 32 bytes. The sequence number is supplied by DCE. The resulting ID field is stored on the IDT in the field defined by the AUTO-ID-FIELD parameter. This field must be defined in the Files section. We recommend defining an ID field with attributes `F,10`. This leaves ample room for an `Auto-Id-Name` and several characters for the sequence number. Since the latter is a base 36 number, it allows for 1.6 million records in 4 characters, 60 million using 5 characters, or up to 3.6 Gig with a 6 character sequence number. The length attribute of the ID field is not limited to 10 bytes. It may be increased when you have a large number of records and/or a long `Auto-Id-Name` prefix. **Note:** Refer to the *IDX Definition / AUTO-ID-FIELD=* section.

Clustering Definition

This section begins with the Clustering-Definition keyword. The fields are as follows:

Field	Description
NAME=	A character string which identifies the `Clustering-Definition`. The name must not match any `Search-Definition` nor `Multi-Search-Definition` names in the same Project. This is a mandatory parameter.
CLUSTERING-ID=	A unique two-character ID prefixed to all cluster numbers generated by this Clustering. This is a mandatory parameter. If the first Clustering definition is used for seeding and any subsequent Clusterings are adding to this seeded Clustering then all these Clusterings should use the same `CLUSTERING-ID`. See the *CLUSTERING-METHOD=SEED* section under User-Job-Definition for more information about seeding.
IDX=	The name of the IDX used by the clustering step. If this parameter is not given, then the default IDX name "kx" followed by the given `CLUSTERING-ID` is assumed. The IDX is defined in the *IDX-definition* section (see the *IDX Definition / NAME=* section.
INDEXES-PATH=	The path for Clustering index files. **Note:** DCE does not support spaces in file or PATH names
FORMATTED-FILE-PATH=	Optional path for the formatted data file `fmt.tmp`. Refer to the *Reformat Input Data* section.
COMMENT=	An optional character string describing this clustering step.
SEARCH-LOGIC= (alias to KEY-LOGIC=)	This parameter describes the logic to be used to generate search ranges to find candidate records from the IDT. It may differ from the `KEY-LOGIC=` used to generate keys for the IDT (as defined in the IDX-Definition). Refer to the *Search Logic* section for details. This is a mandatory parameter.
SCORE-LOGIC	This parameter describes the normal matching logic used to refine the set of candidate records found by the `Search-Logic`. This is a mandatory parameter unless at least one of the other SCORE-LOGIC parameters is specified. Refer to the *Score Logic* section for details.
PRE-SCORE-LOGIC	This optional parameter describes the lightweight matching logic used to refine the set of candidate records found by the Search-Logic. Refer to the *Score Logic* section for details.
KEY-SCORE-LOGIC=	This optional parameter describes the normal matching logic used to refine the set of candidate records found by the `Key-Logic`. Refer to the *Search Logic* section for details.
KEY-PRE-SCORE-LOGIC=	This optional parameter describes the light-weight matching logic used to refine the set of candidate records found by the `Key-Logic`. Refer to the *Search Logic* section for details.
SORTED-FILE-PATH=	Optional path for the temporary sort data file `srt.tmp`.
SORT-WORK1-PATH=, SORT-WORK2-PATH=	DCE may create sort work files when sorting a large result set. These parameters control the placement of these files and override the values possibly given in the `Project-Definition`.

Field	Description
KEY-FIELD=	The name of the field in the database file which is to be used for key generation purposes. This must be a field defined in the `File-Definition`. It is recommended that any use of this keyword is reviewed and converted to use the newer `Field(List of keyfields)` Search-Logic/Key-Logic option. For more details, refer to the *Search Logic* section.
CANDIDATE-SET-SIZE-LIMIT=n	Informs the DCE Search Server to process searches by first building a list of candidate records, eliminating duplicates, and then scoring the remainder. This process usually makes scoring more efficient. `n` specifies the maximum number of unique entries in the list. The default limit is 10000 records. A value of 0 disables this processing.
SCHEDULE=< list of jobs>	Comma-separated list of jobs scheduled for this clustering. The jobs listed must be defined in the `job-definition` sections.
UNMATCHED -FILE=	The name of the Logical-File entity that describes the Unmatched File. This file is created when running a clustering job with the No-New-Clusters option. When this parameter is defined, the records that did not match any existing clusters are written to the Unmatched File. An output view may be used to format the output file.
OPTIONS=	A comma separated list of keywords used to control various search options: - `ADD-NULL-KEY SORTIT` processing is used to sort the input file into preferred key order. If an input record generates a null key and the IDX-Definition option `No-Null-Key` has been specified, the record is not written to the output file and therefore will not be loaded into the IDT. If you wish this record to be loaded but do not want null keys added to the key index, specify the Add-Null-Key option. This is useful if the record will be reindexed later using a different field. - `APPEND` the input file is appended to an existing file in the database. This option is used to merge two or more input files into one clustering database. Normally, the previous clustering information is erased when a file is loaded to the database. - `AUTO-ID` generate unique record source Id in the Id-field. This option is needed if source identifiers (Source Id) will be used to identify records. - `IGNORE-NOTCH-OVERRIDE` Ignore any adjustments made to the match levels by a search client (Relate or DupFinder) that requests a particular Match-Tolerance. The tolerance is honored but the adjustments are ignored. - `DELAY` delay the building of cluster index 2 (used by `POST`); if you use this option and you wish to run a `POST` job, you will need to schedule an `EXTRACT` job to create the index file. - `FORMAT` run the `PRE` job to pre-format the raw input data. This option is needed if a job of type PRE is used. - `PRE-LOAD` run the `LOADIT` job to preload the input data to the database. This option is needed if a job of type `LOADIT` is used. - `SEARCH-NULL-PARTITION` any search for a record containing a Null-Partition value will search all other partitions. Any search for a record with a non-null partition value will search the null-partition as well. Note that the entire partition value must be null for this to work. - `SORT-IN` run the `SORTIT` job to sort the input data. This option is needed if a job of type `SORTIT` is used. - `TRUNCATE-SET` modifies the behavior of `CANDIDATE-SET-SIZE-LIMIT`. Searches normally continue until all candidates have been considered. `Truncate-Set` will terminate the search once the candidate set is full, thereby limiting the number of candidates that will be considered.

Search Definition

A Search Definition is used to define parameters for a search executed by the Search Server. Search Definitions are used exclusively by the Batch Relate or Batch DupFinder Utilities. It begins with the SEARCH-DEFINITION keyword.

Field	Description
NAME=	This is a character string that identifies the Search-Definition. This is a mandatory parameter. The name must not match any `Clustering-Definition` nor `Multi-Search Definition` names in the same Project.
IDX=	This is a character string that identifies the IDT to be searched. This is a mandatory parameter.
COMMENT=	This is an optional text field that is used to describe the Search's purpose.
SORT-WORK1-PATH=, SORT-WORK2-PATH=	DCE may create sort work files when sorting a large result set. These parameters control the placement of these files and override the value possibly given in the `Project-Definition`. **Note:** DCE does not support spaces in file or PATH names.
SEARCH-LOGIC=	This parameter describes the Search-Logic to be used to generate search ranges to find candidate records from the IDT. It may differ from the Key-Logic used to generate keys for the IDT (as defined in the `IDX-Definition`). Refer to the *Search Logic* section for details. This is a mandatory parameter.
SCORE-LOGIC=	This parameter describes the normal matching logic used to refine the set of candidate records found by the `Search-Logic`. This is a mandatory parameter unless at least one of the other SCORE-LOGIC parameters is specified. Refer to the *Score Logic* section for details.
PRE-SCORE-LOGIC=	This optional parameter describes the lightweight matching logic used to refine the set of candidate records found by the Search-Logic. Refer to the *Score Logic* section for details.
KEY-SCORE-LOGIC=	This optional parameter describes the normal matching logic used to refine the set of candidate records found by the `Key-Logic`. Refer to the *Search Logic* section for details.
KEY-PRE-SCORE-LOGIC=	This optional parameter describes the light-weight matching logic used to refine the set of candidate records found by the `Key-Logic`. Refer to the *Search Logic* section for details.

Field	Description
SORT=	A comma separated list of keywords used to control the sort order of records returned by the search. Multiple sort keys are permitted. The keys will be used in the order of definition. If not specified, the records will be sorted using a primary descending sort on Score, and a secondary ascending sort using the whole record. `Memory(recs)` - The maximum number of records to sort in memory. If the set contains more than `recs` records, the sort will use temporary work files on disk. The default is 1000 records. `Field(fieldname)` - The name of the field to be used as the sort key. In addition to IDT field names, you may specify the following values: - `sort_on_score` will sort on the Score - `sort_on_record` will sort on the whole record. `Type(type)` - The type of the sort for the previously defined key. Valid types are: - A - Ascending - D - Descending `Format(format)` The format of the previously defined key. Valid formats are: - `FORMAT_TEXT` - `FORMAT_SHORT` native signed short - `FORMAT_USHORT` native unsigned short - `FORMAT_LONG` native long - `FORMAT_ULONG` native unsigned long - `FORMAT_FLOAT` native float - `FORMAT_DOUBLE` native long float - `FORMAT_NN_SHORT` non-native short - `FORMAT_NN_USHORT` non-native unsigned short - `FORMAT_NN_LONG` non-native long - `FORMAT_NN_ULONG` non-native unsigned long

Field	Description
CANDIDATE-SET-SIZE-LIMIT=n	Informs the DCE Search Server to optimize the matching process by eliminating any duplicate candidates that have already been processed. `n` specifies the maximum number of unique entries in the list. The default limit is 10000 records. A value of 0 disables this processing. If there are more than `n` unique candidates, only the first `n` duplicates are removed. Any candidates, which can not fit into the list, may be processed several times, and if accepted by matching, added to the final set several times. The TRUNCATE-SET option will terminate the search for candidates once the list becomes full. It is used to prevent very wide searches. However, if a search is terminated prematurely there is no guarantee that any of the candidates will be accepted and/or the best candidates have been found.
OPTIONS=	A comma separated list of keywords used to control various search options: - `FIRST` stop on first accepted match. Used for specialized applications where any acceptable match should terminate the search process. - `HIDDEN` prevents this search from being listed by the Search Clients such as Relate and DupFinder if the search is not designed to be used independently (ie. it should only be used as part of a Multi- Search). - `IGNORE-NOTCH-OVERRIDE` Ignore any adjustments made to the match levels by a search client (`Relate` or `DupFinder`) that requests a particular Match-Tolerance. The tolerance is honored but the adjustments are ignored. - `SEARCH-NULL-PARTITION` any search for a record containing a Null-Partition value will search all other partitions. Any search for a record with a non-null partition value will search the null-partition as well. Note that the entire partition value must be null for this to work. - `TRUNCATE-SET` modifies the behavior of `CANDIDATE-SET-SIZE-LIMIT`. Searches normally continue until all candidates have been considered. `TRUNCATE-SET` will terminate the selection of candidates once the candidate set is full, thereby limiting the number of candidates that will be considered. - `UNIQUE-KEYS` specifies that no duplicate sort keys will be returned. Sort keys are defined with the SORT= keyword. - `UNMATCHED-STATS` Used when an output view contains performance counters (e.g. the number of candidates) and the search result set is empty. When enabled, the Search Server will return a dummy IDT record filled with asterisks as a vehicle to return the performance counters. Otherwise statistics are not returned when the set is empty.

Multi-Search Definition

A `MULTI-SEARCH-DEFINITION` is used to define parameters for a cascade of searches to be executed by the DCE Search Server. Multi-Search Definitions are used exclusively by the `Batch Relate` or `Batch DupFinder` Utilities. It begins with the MULTI-SEARCH-DEFINITION keyword.

Field	Description
NAME=	This is a character string that identifies the Multi-Search. It is a mandatory parameter. The name must not match any `Clustering-Definition` nor `Search-Definition` names in the same Project.
COMMENT=	This is a text field that is used to describe the Multi-Search's purpose.
SEARCH-LIST=	This is a string that contains the list of searches to perform. The searches should be separated by commas and enclosed in double quotes. All searches must be run against the same IDT. Maximum of 16 searches can be specified. When the statistical output view field `IDS-MS-SEARCH-NUM` is used, the returned result refers to the search names in this list so that the returned value 1 means the first search in this list, value 2 means the second search in this list etc. See the *Statistical Fields* section for more information.

Field	Description
IDT-NAME=	This is a character string that identifies the IDT over which the Multi-Search is to be performed. It is a mandatory parameter.
SOURCE-DEDUP-MAX=	Multi-Search processing can keep track of the candidate records processed in a particular search to avoid reprocessing them when they are retrieved as candidates in a later search (in the `Search-List`). This is useful and appropriate only if all searches use the same `Score-Logic`. This parameter specifies the number of records that will be remembered. The default value of 0 disables this feature.
DEDUP-PROGRESS=	A parameter that controls the deduplication process. The `DupFinder` process treats each IDT record as a search record instead of reading search records provided by a client. Normally the search process does not return control to the client until a duplicate is found. This may take a long time if the are very few duplicates in the IDT. This parameter is used to specify the maximum number of IDT records to be processed before returning control to the client process. This gives the client the opportunity to write progress messages. The default value of 0 disables this feature.
OPTIONS=	A comma separated list of keywords used to control various search options: - `FULL-SEARCH` specifies that a Multi-Search is to process all searches in the list rather then returning on the first search that returns some data.

Multi-Clustering-Definition

Use multi-clustering definition to define parameters for a cascade of clusters. The Data Clustering Engine clustering process uses the multi-clustering definition to run the cascade of clusters. The clustering process starts with the multi-clustering definition keyword.

The following table lists the components of a multi-clustering definition:

Field	Description
NAME=	A unique character string that identifies the multi-clustering definition. The name field is mandatory and must not match any clustering definition nor search definition names in the same project.
CLUSTERING-ID=	A unique two-character ID prefixed to all cluster numbers. The `CLUSTERING-ID` field is mandatory. The clustering process verifies the Cluster ID value with cluster-id definition in the `CLUSTERING-LIST` field.
COMMENT=	Description of the purpose of the multi-clustering definition.
CLUSTERING-LIST=	The string that contains the list of clustering to perform. Separate the clusters with commas and enclose within double quotes. Run all clustering against the same Identity Table. The maximum search value is 16 searches.
IDT-NAME=	A character string that identifies the Identity Table over which the system should perform multi-clustering. IDT-NAME field is mandatory.

Field	Description
OPTIONS=	A list of keywords used to control various search options. Separate each search option with a comma. `FULL-SEARCH` option specifies that a multi-clustering definition is to process all clusters in the list else it returns on the first cluster that returns some data. When you enable the `FULL-SEARCH` option, it executes all search defined in the clustering-list in parallel.
SCHEDULE=<list of jobs>	A comma separated list of jobs in queue for multi-clustering. The jobs in the list must be included in the job definition section. The `list of jobs` should contain jobs related to clustering and post report processing.

Note: The jobs defined in multi-clustering definition should not schedule to perform a LOAD-IDT job.

The following definition lists a sample multi-clustering definition:

```
MULTI-CLUSTERING-DEFINITION
*======================
NAME=                          multi-clustering
CLUSTERING-ID=                 AA
IDT-NAME=                      IDT-100
CLUSTERING-LIST=               clustering-name,
                               clustering-address,
                               clustering-company
Options=                       FULL-SEARCH
SCHEDULE=                      job-cluster,
                               job-post-all
*
job-definition
*=============
NAME=                          job-cluster
TYPE=                          cluster
CLUSTERING-METHOD=             Merge
*
job-definition
*=============
NAME=                          job-post-all
TYPE=                          post
FILE=                          lf-post-all
OUTPUT-OPTIONS=                Trim, CR
*
```

User-Job-Definition

A User-Job-Definition is the SDF equivalent of a Console Job. Prior to version 2.6, User Jobs could only be defined using the Console's **Jobs** > **Edit** > **New Job** dialog. It is envisioned that customers will continue to use the GUI to define Jobs. However, in order to facilitate the transfer of pre-defined jobs between Dev, Test, QA and Prod environments a new mechanism was needed to export the definition from the Rulebase into an SDF, and the SDF was enhanced by adding syntax for defining User-Jobs.

Note: User-Jobs are a new experimental feature of DCE. They are partially documented here to facilitate user experimentation and feedback. The keywords and/or syntax may change in a future release.

Jobs are exported from the Rulebase into SDF format using **System** > **Export** > **SDF**. The parameters associated with a Job Step mirror the parameter names in the equivalent GUI dialog used to create it. The easiest way to get started is to define a Job using the Console and export it to an SDF.

A `User-Job-Definition` contains two parameters:

Field	Description
NAME=	Defines the name of the User-Job. All subordinate User-Step-Definitions quote this name to associate themselves with the User-Job-Definition.
COMMENT=	This is an optional text field that is used to describe User-Job's purpose.

User-Step-Definition

Each `User-Job-Definition` is associated with one or more `User-Step-Definitions`. They are equivalent to Job Steps added with the **Add Step** button in the Console and contain the following parameters:

Field	Description
COMMENT=	This is a text field that is used to describe step's purpose.
JOB=	This is the name of the User-Job-Definition that this step belongs to.
NUMBER=	This field is used to order steps within a User-Job-Definition. `NUMBER` is a printable numeric value starting from 0. That is, the first step has `NUMBER = 0`. There must be no gaps in the numbering of steps.
NAME=	This is the type of Job step. A list of valid types is visible in the Console dialog when you click **New Step**. Names containing spaces should be enclosed in quotes (").

TYPE=	This is the type of step. A list of valid job types and their associated parameters can be generated by running
	`%SSABIN%\pdf -ddce reportFileName`
	or
	`$SSABIN/pdf -ddce reportFileName`

PARAMETERS=	This is a list of parameters and values required by the step (if any). A list of valid job types and their associated parameters can be generated by running
	`%SSABIN%\pdf -ddce reportFileName`
	or
	`$SSABIN/pdf -ddce reportFileName`

For example:

```
user-job-definition
*==================
NAME=                          RunClustering1AndDupFinder
*
user-step-definition
*===================
COMMENT=                       "First step"
JOB=                           RunClustering1AndDupFinder
NUMBER=                        0
NAME=                          "Reinitialize DB"
TYPE=                          "Reinitialize DB"
*
user-step-definition
*===================
```

```
COMMENT=                        "Second step"
JOB=                            RunClustering1AndDupFinder
NUMBER=                         1
NAME=                           "Run Clustering 1"
TYPE=                           "Run Clustering"
PARAMETERS=                     ("Clustering Definition",
                                        clustering-1)

*
user-step-definition
*==================
COMMENT=                        "Third step"
JOB=                            RunClustering1AndDupFinder
NUMBER=                         2
NAME=                           "Run DupFinder"
TYPE=                           DupFinder
PARAMETERS=                     ("Output File","dup00.rep"),
                                ("Search Definition",
                                         clustering-1),
                                ("Output Format",0),
                                ("Append New Line",true),
                                ("Trim Trailing Blanks",true),
                                ("Return Search Records Only",
                                        false),
                                ("Remove Search Record",false)
```

The Job Definition

This section begins with the Job-Definition keyword. The fields are as follows:

Field	Description
NAME=	A character string identifying the job. This is a mandatory parameter.
COMMENT=	This is a text field that is used to describe the Job's purpose.
IDX-LIST=	This is a comma-separated list of IDX names used in conjunction with the Load-All-Indexes option to limit the number of IDXs to be loaded. Normally Load-All-Indexes means that all IDXs that have been defined are to be loaded.
FILE=	A parameter used to define name of the Logical-File entity which describes either an input or output file to be used by this job.
TYPE={PRE|SORTIT| LOADIT|CLUSTER| EXTRACT|POST}	A character string that describes the type of job. Refer to the *Clustering Suite* section in the Introduction chapter for more details. This is a mandatory parameter.

Field	Description
CLUSTERING-METHOD=method	Specifies how to assign records to clusters for a job of type `CLUSTER`. This setting is ignored for other job types. The parameter method is one of the following: **BEST** A new record is added to the cluster that contains the best matching record, or is allocated to a new cluster if its score does not reach the specified minimum for a successful (i.e. accepted) match. Best is the default Clustering-Method Option. **MERGE** All clusters that the record is successfully matched with (i.e. are accepted) are merged into a single cluster and the new record is added to that cluster. **SEED** Each record is allocated to a separate cluster. For example, if the only requirement is to discover the records in one file which match to records in another file, then the larger of the two files could be `SEED`ed first, then the second file `APPEND`ed to that file using the NONEWCLUSTERS option. **PRE-CLUSTERED[(FieldName[,Grouped,Ordered])]** The input file is pre-clustered. Each input record contains a field with a cluster number in it. This `FieldName` defaults to `cluster`, although you may nominate any `FieldName` from your view definition. Unless the additional optional keywords Grouped and Ordered are used, the field must be defined with a format of `I,4` on the Database file, although it may have any compatible format on the input view. Also, the original cluster numbers will be translated to new cluster numbers when loaded into the database. Consequently, if two or more input files contain records with the same cluster numbers, these clusters will not be maintained when loaded into the database. Each file will be allocated to a different range of cluster numbers. **Note:** Cluster number 0 is not allowed. An example of using `PRE-CLUSTERED` is when the input file already has duplicate id-numbers, which define groupings of related records. This file may then be re-clustered using a different key field, and the NO-ADD option. If the additional optional keywords Grouped and Ordered are used, the pre-clustered field can be any size and any format. The input file must have all records with the same ID value consecutive (`Grouped`) and in ascending order (`Ordered`). Currently the Grouped option cannot be used without the Ordered option. We will later introduce internal sorting to make this ordering optional. **MANY** A record may become a member of multiple clusters. This option is not compatible with the MERGE option. For example, if the requirement is to populate a prospect file with do not mail or fraud records, the prospect file could be `SEED`ed, then the do not mail or fraud records could be clustered to this file using the MANY option. **FIRST** The first candidate that achieves an Accepted score is accepted into a cluster. This option introduces an order dependency. It is designed to be used only when the resulting clusters are to be discarded and the unmatched file contains the records of interest.

Field	Description
CHECKPOINT-TIME=n[s|m|h|d]	This parameter informs the Data Clustering Engine to enter a Wait state after clustering records for `n` seconds/minutes/hours/days. `n` is assumed to have units of seconds if it is not qualified by the optional s, m, h or d unit parameter. Refer to the *Stopping and Restarting Clustering* section for more information on how to use this parameter.
STATUS-TIME=n[s|m|h|d]	This parameter informs the Data Clustering Engine to write a status report after clustering records for `n` seconds/minutes/hours/days. `n` is assumed to have units of seconds if it is not qualified by the optional s, m, h or d unit parameter.
INPUT-SELECT=n, INPUT-SELECT=[Count(n),] [Skip(n),] [Sample(n)]	This parameter is used to define input file processing options. When specified in the first form above, the number `n` is treated as the number of records to be read from the input file. An equivalent method of specifying this is `Count(n)`. The value `n` must be a positive non-zero number. You may skip some records before processing begins by specifying `Skip(n)`. You may also process every nth record by specifying `Sample(n)`. Note that the `INPUT-SELECT` statement is ignored by the Cluster step if the data has been preloaded. In this case you can use the `INPUT-SELECT` statement in the `LOADIT` step.
INPUT-HEADER=	Describes the number of bytes to ignore at the start of the input file. This is useful for some types of files that contain a fixed length header before the actual data records.

Field	Description
OPTIONS=	A comma separated list of option keywords for the job: **INPUT-APPEND** This causes the `PRE` job to append its output to an existing `PRE` output file. This allows running multiple `PRE` steps before the `SORTIT` step. **NO-NEW-CLUSTERS** if a record does not successfully match any existing records, do not create a new cluster for it. i.e. do not create any clustering relationship. In this case, the unmatched records can be written to the file specified by the UNMATCHED-FILE Clustering parameter. **MATCH-ALL-MEMBERS** Match the record against all members of the cluster, not just the voting members. **LOAD-ALL-INDEXES** Instructs `LOADIT` to load all indexes declared in the Project definition file. This is useful when there is only one clustering loading a file (seeding) and more than one search/ clustering has been defined using different indexes. **RE-INDEX** Instructs `LOADIT` to read records from an existing database file and generate a new key index instead of reading from an input file. This is useful when there is a desire to re-cluster the file using a new key field to improve the reliability of the clusters. Refer to the *How To Re-Cluster Data* section for more details. **Note:** If the data used for the new index is available during the initial load (first clustering job), the option LOAD-ALL-INDEXES can be used to load all the indexes during the initial load. **NO-ADD** Used to recluster records which were loaded as preclustered, or previously clustered. `NO-ADD` prevents data records from being added to the database. Refer to the *How To Re-Cluster Data* section on for more details. **USE-ATTRIBUTES** Honors the voting attribute. By default this option is turned off, except when Clustering-Method, **Many** is used without the NO-ADD option, in which case this option is turned on. Refer to the *Voting Attribute* section for more information. **SET-ALL-VOTE** All members that are added can vote. This is the default, except when Clustering- Method, `Many` is used without the NO-ADD option, in which case `SET-VOTE-NONE` is the default. Refer to the *Voting Attribute* section for more information. **SET-NONE-VOTE** All members that are added can not vote. This requires that there already are existing cluster members that can vote (from a previous clustering). If not, the result would be that all records would become single-member clusters. Any new clusters created will stay as single-member clusters. Refer to the *Voting Attribute* section for more information.

Field	Description
	SET-HEADERS-VOTE Only the founding member of the cluster can vote (one per cluster). Refer to the *Voting Attribute* section for more information. **STATUS-APPEND** Append messages to the Clustering Status File instead of overwriting it. **REVERSE-SORTIN** Reverse the sort order of keys in the SORTIT job. PARTITION data is not reversed. `KEY-DATA` is not used by the sort process. **Note:** This option is only available for Beta testing. It may or may not be present in future releases. **NO-PARTITIONS-STATS** Disable logging of Partitions statistics. If you have a very large number of partitions then it is recommended since the logging will slow the process significantly. **THREADS(#)** Refer to the Utilizing multiple CPUs section for more details.
CANDIDATE-SET-SIZE-LIMIT=n	Informs the `CLUSTER` step to process searches by building a list of candidate records, eliminating duplicates, and then scoring the remainder. `n` specifies the maximum number of unique entries in the list. The default limit is 10000 records. A value of 0 disables this processing. Any candidates that do not fit in the list generate an Audit Trail record of type `Overflow`. This process makes scoring more efficient, when candidates are found more than once. However, it can affect the clustering results if the Clustering-Method is sensitive to the order in which records are scored. Example, the BEST method will select the record with the best score, but if two or more records achieve the best score, then the first is selected. As deduping can reorder the records, a different record might be selected and the clustering result may differ over two otherwise identical runs. The TRUNCATE-SET option will terminate the search for candidates once the list becomes full. It is used to prevent very wide searches. However, if a search is terminated prematurely there is no guarantee that any of the candidates will be accepted and/or the best candidates have been found.
CANDIDATE-SET-WARNING-LEVEL=n	This specifies a threshold value. If the set of candidate records is greater than or equal to this limit `n`, an Audit Trail record (type `SetWarning`) is written. The default value is one quarter of the `CANDIDATE-SET-SIZE-LIMIT`.

Field	Description
CANDIDATE-SET-REPORT-LIMIT=n	The cluster step will tabulate the number of records in each candidate set. This parameter n determines the size of the biggest set for which discrete counts will be maintained. At the end of the `CLUSTER` job, a histogram will be displayed (entitled Histogram: ranges - candidates count). The default value is equal to the `CANDIDATE-SET-SIZE-LIMIT`.
OUTPUT-OPTIONS=	A comma separated list of options for the `POST` job. **Only-Singles** Only report single member clusters. **Only-Plurals** Only report clusters with more than one member. **Indent** Blank out the Cluster ID except for the first member in each cluster. The default is to fill in DCE on all cluster members. **Trim** Remove trailing blanks from output records. **CR** Add carriage return to the end of output records. **Report** This enables the following combination of options: Indent, Trim, CR. **Layout** Write the view description used to generate the report into the report header. **Voting** Only report Voting records. To report only Non-Voting, use `--Voting`. Default is to report all members.

The following table shows the options that are applicable to each job type.

	pre	sortit	loadit	cluster	extract	post
NAME	✓	✓	✓	✓	✓	✓
COMMENT	✓	✓	✓	✓	✓	✓
TYPE	✓	✓	✓	✓	✓	✓
FILE	✓	✓	✓	✓		✓
CLUSTERING-METHOD				✓		

	pre	sortit	loadit	cluster	extract	post
CHECKPOINT-TIME				✓		
STATUS-TIME				✓		
INPUT-SELECT	✓	✓	✓			
INPUT-HEADER	✓	✓	✓			
OPT=INPUT-APPEND	✓					
OPT=NO-NEW-CLUSTERS				✓		
OPT=RE-INDEX			✓			
OPT=NO-ADD				✓		
OPT=USE-ATTRIBUTES				✓		
OPT=SET-ALL-VOTE				✓		
OPT=SET-NONE-VOTE				✓		
OPT=SET-HEADERS-VOTE				✓		
OPT=STATUS-APPEND				✓		
OPT=REVERSE-SORTIN		✓				
CANDIDATE-SET-*				✓		
OUTPUT-OPTIONS	✓					✓

Search Logic

`Search-Logic` or `KEY-LOGIC` defined in the `Clustering-Definition` and/or `Search-Definition` instructs DCE how to build search ranges to access the IDT using the index.

There are two types of Search-Logic (Key-Logic):

- SSA
- User

Search Logic (SSA)

Search-logic (Key-Logic) facilities are provided by SSA-NAME3 Population Rules.

```
KEY|SEARCH-LOGIC=SSA,
System(system),
Population(population),
Controls(controls),
Field(keyfield_list),
[Null-Field(null-field-value)],
[Null-Key(null-key-value)]
```

where

system

Defines the System Name of the SSA-NAME Population Rule File. The `system` is a sub-directory of the population rules directory. The location of the population rules directory is defined by the environment variable *SSAPR* .

population

Defines the name of the Population file which is located in the `system` directory. The Population chosen will depend on the data being processed.

controls

Used to describe the data type of the Key Field data and to control the thoroughness of the search

Note: For more details on the `Controls`, refer to the *SSA-NAME3 API REFERENCE GUIDE* under the **ssa_get_ranges** heading within the *Controls* section.

keyfield_list

This is a comma separated list of IDT fields to be used to generate keys or ranges. If more than one field is provided, the **keyfield_list** must be enclosed in quotes (").

null-field-value

An optional parameter that defines the null value for the `Field`. Records with a null value in their key field can be ignored by using the NO-NULL-FIELD option to prevent them from being stored in the IDT. The default value is spaces (blanks).

For example,

```
Null-Field("unknown")
```

null-key-value

An optional parameter that defines the value of a null key. Records with a null key can be ignored by using the NO-NULL-KEY option to prevent them from being stored in the IDX. The default value is "K$$$$ $$$".

For example,

```
Null-Key("K$$$$$$$")
```

The above key value will be generated from a name containing null data or composed of only delete words.

Controls

A `Key-Logic` in an `IDX-Definition` controls the generation of keys. Therefore the Controls should specify the `KEY_LEVEL`. For example,

```
KEY-LOGIC=SSA,
          System(system), Population(population),
          Controls("FIELD=fieldname KEY_LEVEL=Standard"),
          Field(keyfield)
```

A Search-Logic in a Search-Definition controls the generation of search ranges. Therefore the Controls should specify a `SEARCH_LEVEL`. For example,

```
SEARCH-LOGIC=SSA,
             System(system), Population(population),
             Controls("FIELD=fieldname SEARCH_LEVEL=Typical"),
             Field(keyfield)
```

Repeating Key Fields

The Field parameter of the Key- or Search-Logic can be used to generate keys or ranges for multiple fields (of the same type). This is accomplished by specifying a list of fields.

For example,

```
IDX-Definition
*=============
KEY-LOGIC=SSA,
          System(system), Population(population),
          Controls("FIELD=fieldname KEY_LEVEL=Standard"),
          Field("field_1,field_2,field_3")
```

will generate keys for all three name fields and store them in the IDX.

For search:

```
SEARCH-LOGIC=SSA,
             System(system), Population(population),
             Controls("FIELD=fieldname SEARCH_LEVEL=Standard"),
             Field("field_1,field_2,field_3")
```

will generate search ranges for all three name fields.

For Matching:

```
SCORE-LOGIC=SSA,
            System(system), Population(population),
            Controls("PURPOSE=purpose_name MATCH_LEVEL=Loose"),
            Matching-Fields("fieldn ame_1:Person_Name",
            "fieldname_2:Person_Name","textitfieldname_3:Person_Name")
```

Will apply repeat logic to all three name fields.

When indexing a Group of repeating fields (or flattened fields), you must list each individual field name. For example, the following source definition

```
test.person.nameName [5] C(20)
```

generates a Group in the File-Definition of this IDT, which is subsequently expanded to the following list of field names:

```
FIELD=Name,   C, 20
FIELD=Name_2, C, 20
FIELD=Name_3, C, 20
```

```
FIELD=Name_4, C, 20
FIELD=Name_5, C, 20
```

Therefore a Key-Logic that indexes these fields must list each individual field:

```
KEY-LOGIC=SSA,
          System(default), Population(test),
          Controls("FIELD=KeyFieldName KEY_LEVEL=Standard"),
          Field("Name,Name_2,Name_3,Name_4,Name_5")
```

A shorthand method of specifying repeating fields exists and takes the form of:

```
field [ * | {x | y-z}, ... ]
```

Some examples of this notation are (assuming a maximum of 5 occurrences):

```
Name[1-5]  = Name,Name_2,Name_3,Name_4,Name_5
Name[2,3]  = Name[2-3] = Name_2,Name_3
Name[1,4-5]= Name,Name_4,Name_5
Name[5]    = Name_5
Name[*]    = Name,Name_2,Name_3,Name_4,Name_5
```

Therefore some of the ways the above Key-Logic example could also be written are:

```
KEY-LOGIC=SSA,
          System(default), Population(test),
          Controls("FIELD=KeyFieldName KEY_LEVEL=Standard"),
          Field("Name[*]")
KEY-LOGIC=SSA,
          System(default), Population(test),
          Controls("FIELD=KeyFieldName KEY_LEVEL=Standard"),
          Field("Name[1-5]")
KEY-LOGIC=SSA,
          System(default), Population(test),
          Controls("FIELD=KeyFieldName KEY_LEVEL=Standard"),
          Field("Name[1,2,3,4,5]")
```

Key Logic (User)

For `User` logic the value of the key field `fld` is used as the key.

```
KEY-LOGIC=User, Field(KeyFieldName)
```

Score Logic

Score Logic is defined in the `Clustering-Definition`. It specifies how the DCE Search Server will select records to be returned from the set of candidates. Score-logic facilities are provided by SSA-NAME3 Population Rules.

The score logic has four possible steps:

```
KEY-SCORE-LOGIC
KEY-PRE-SCORE-LOGIC
PRE-SCORE-LOGIC
SCORE-LOGIC
```

At least one score-logic step must be defined.

`KEY-PRE-SCORE-LOGIC` is used to define an optional "light weight" scoring scheme that is processed before the `KEY-SCORE-LOGIC`. Its purpose is to make a fast and inexpensive decision as to whether or not the more costly `KEY-SCORE-LOGIC` should be called to process the current record.

KEY-SCORE-LOGIC is used to define a scoring scheme to score KEY-DATA. Refer to the *Key Data/Key-Score-Logic* section for details.

PRE-SCORE-LOGIC is used to define an optional "light weight" scoring scheme that is processed before the SCORE-LOGIC. Its purpose is to make a fast and inexpensive decision as to whether or not the more costly SCORE-LOGIC should be called to process the current record.

All the SCORE-LOGIC keywords share the following syntax:

```
SCORE-LOGIC = <type>,<parm-list>
```

where

<type>

This is SSA (currently only SSA Score-Logic is supported).

<parm-list>

This is a comma-separated list of parameters.

Parameter list:

```
System(system)
Population(population)
Controls(controls)
Matching-Fields(field-list)
```

where

system

Defines the System Name of the Population Rules.

population

Defines the population name. The population chosen will depend on the data being processed.

controls

Defines the controls to be used for the Score-Logic. Controls should specify the desired PURPOSE and MATCH_LEVEL.

Note: For more details on the Controls, refer to the *SSA-NAME3 API REFERENCE GUIDE* under the **ssa_match** heading within the Controls section.

Matching-Fields

Holds the list of fields and their data type used by the matching routine.

field-list

This is a comma-separated list of the form **field_name:data_type**where

field_name

This is the name of a field in the IDT.

data_type

This is the data type that this field represents, as defined in the Population Rules.

The field_list defines which IDT fields will be used for matching and the type of data they represent. Matching will use repeat logic when two or more fields of the same data_type are specified. A run-time error will occur if a data_type defined as mandatory for the PURPOSE has not been specified in the field_list. Optional data_types may be omitted.

For example,

```
SCORE-LOGIC=SSA,
            System(system), Population(population),
```

```
Controls ("PURPOSE=purpose_name MATCH_LEVEL=Loose"),
Matching-Fields("fieldname:Person_Name",
                "field_date:Date",
                "field_sex:Attribute1",
                "field_postal_area:Postal_Area")
```

FILES and VIEWS Sections

These sections are used to define files and views. Files describe the layout of DCE IDT created from flat-file input. Views are used to:

- describe the layout of flat-file input data
- transform flat-file input data
- format the output from the DCE Search Server

Data Source

When loading DCE data from User Source Tables, the Files Definition and View Definition files are created automatically from the User Source Table Definition. The name of the generated Files definition is the same as the IDT name in the `CREATE_IDT` clause. The View names(s) are created by concatenating the IDT name with a sequence number starting from 1.

When loading DCE Tables from flat (external) files, the File and View definitions must be hand-coded. The File definition describes the layout of the IDT, while the View Definition describes the layout of the input file.

File Definition

A set of one or more File Definitions begins with the section heading:

```
section: Files
```

A File Definition begins with the FILE-DEFINITION keyword. It is used to describe the layout of the IDT. Although it is possible to specify more than one File Definition, only one definition can be in effect for each IDT.

A File Definition contains one parameter which is unique to File Definitions. This is the ID= parameter. It is mandatory and is used to allocate a unique number to the IDT (for internal use). Specify a positive, non-zero number that is less than 2000. File numbers greater than 1999 are reserved for internal use.

View Definition

A set of one or more View Definitions begins with the section heading:

```
section: Views
```

A View Definition begins with the VIEW-DEFINITION keyword. As many views as necessary may be defined.

A view normally consists of a subset of fields taken from a File Definition. It may contain additional fields that are added by user-defined Transformations (Refer to the *Transformations* section for more details).

Syntax

The definition starts with a name as described above. It is followed by:

Field	Description
NAME=	This character string names the file or view. It may be up to 32 bytes long.
ID=	This parameter is only specified for File Definitions; see the *File Definition* section above.
FIELD=name,format,length	The parameter is used to define the fields that comprise the File or View. The maximum number of fields is platform specific. The order of `FIELD` definition statements determines the physical ordering of fields on the database record and/or view. **name** A character string that names a field. It may be up to 32 bytes in length. **format** The format of the field. This may be defined using a pre-defined format, or as a customized format. See the *Field Formats* section below. **length** The length of the field in bytes. The maximum length is platform specific. Refer to the *Limitations* section for details.

Field Formats

The format of a field may be specified in one of two ways:

- a pre-defined format
- a customized format definition

A pre-defined format is a shorthand way of selecting a pre-defined group of field attributes. It is selected by specifying the pre-defined name for the `<format>` value.

A customized format definition is used when no pre-defined format exists for the combination of attributes that you desire. You may combine various field attributes to your liking (as long as they do not conflict).

The following table lists the pre-defined formats and their attributes:

Format	Compression	Data	Base	Conv	Just	Filler
F	Fixed	Text	N/A	No	Right	' '
C	Variable	Text	N/A	No	Left	' '
V[1]	V-type	Text	N/A	No	Left	' '
B[2]	Variable	Binary	N/A	No	Left	0x00
I[3]	Variable	Integer	0	Yes	Right	0x00
G	Fixed	Integer	0	Yes	Right	0x00

Format	Compression	Data	Base	Conv	Just	Filler
N[4]	Variable	Numeric	10	Yes	Right	'0'
X[4]	Variable	Numeric	16	Yes	Right	'0'
Z[4]	Variable	Numeric	36	Yes	Right	'0'
R[4,5]	Variable	Numeric	10	Yes	Right	' '

[1] V-type will compress multiple embedded blanks from the input view into one blank in the target field. Therefore it should only be defined in a File-Definition. It has no effect in a View-Definition.

[2]The Binary data type can hold any unstructured data. It is not necessarily a base 2 number.

[3]Valid integer lengths are 1, 2, 3 or 4 bytes.

[4]The Numeric data type is the printed representation of an unsigned (positive) number.

[5]R format is equivalent to N, but with leading spaces instead of zeroes.

Compression

The compression attribute determines how a field is compressed/decompressed when writing to/reading from the database. Fixed compression is effectively no compression at all; the field is stored/retrieved without any compression or decompression. Variable compression means that the data will be compressed when written to the database and decompressed (to match the view definition) when read from the database. The filler character and justification determine which character is stripped /added from the record and from which end.

V-type compression will compress multiple adjacent blanks into a single blank. This can occur anywhere within the field. Note that this process is irreversible. Decompression will only add blanks at one end of the field.

Filler Character/Justification

The decompression will add the filler character to the field to pad it to the necessary length.

The compression logic will remove the filler character from either the left or right end of the field (depending on justification), until no more filler characters are left or the field is empty. The decompression will add the filler character to the field to pad it to the necessary length (as specified in the view). If the field is left justified, the truncation/padding occurs on the right. If the field is right justified, the truncation/padding occurs on the left side of the field.

Data Type

Four data types are supported:

Text	Character data
Binary	Binary (unstructured) data
Integer	1, 2, 3 or 4 byte unsigned integers
Numeric	Fields containing printable numeric digits '0' to '9', and 'a' to 'z' (when using base 36)

Base/Base Conversion

The Integer and Numeric data types support base conversion. Ie. the view may request a base which differs from the base of the stored data. Base conversion is only possible for bases 2 through 36.

Customized Format Definition Syntax

A customized format definition has the following syntax.

```
format=( [predefined fmt,] format specifier, ...)
```

You may base your definition on a predefined format and override certain attributes, or fully define a new custom format using format specifiers.

The `format specifier` is one of the following:

Fixed	fixed compression
Text	text data
Ljust	left justification
Rjust	right justification
Filler(<char>)	filler character
Base(nn)	base nn

Here is an example of a field definition with explicit format:

```
Attributes, (Base(2), Filler(0)), 16
```

This will format a numeric field into 16 characters in base 2 (binary base). Each position will be either 'zero' or 'one' and leading zeroes (the filler) will extend the value on the left if it is shorter that 16 characters. If we want the value as four hexadecimal digits instead then we should use this format:

```
Attributes, (Base(16), Filler(0)), 4
```

Which is the same as the short form (in the table above):

```
Attributes, X, 4
```

Transformations

Fields within a View Definition may specify an optional transformation. Only Transformations that manipulate the source data are supported in output.

A transformation definition follows the field's length attribute and has the following format:

```
Xform(transform [, parameter-list])
```

where `transform` is one of the following:

APPEND

Appends the current field with the target field by removing trailing spaces from the target field and adding the current field. (input only)

APPEND-STRING

Appends a character string by removing all trailing space and adding the string. The `parameter-list` contains the character string. (input only)

CONCATENATE

Concatenate the current field with the target field by removing trailing spaces from the target field, adding a space, and adding the current field. (input only)

FILL

Fills the target field with a given pattern. The parameter-list contains the character string pattern.

INSERT-CONSTANT

Inserts a character string into a field. The parameter-list contains the character string.

INSERT-FIELD

Inserts the current field to the target field at a specified fixed offset. (input only)

LOWERCASE

Convert a character field to lower case.

UPPERCASE

Convert a character field to upper case.

USER-EXIT

Calls a user-specified routine to perform the transformation. Refer to the Transformation User-Exit section for more details.

Transformation can be applied to both input and output processes.

Multiple transformations can be defined on a single field. They are processed in the order of specification (ie left to right).

A source field can populate multiple target fields by using the following syntax:

```
FIELD=name,format,length,Xform(transform[,parameter-list]),
                         Xform(transform[,parameter-list])
```

Example:

```
FIELD=Lname, C, 15, Xform(INSERT-FIELD, "target LNAME1"),
                    Xform(INSERT-FIELD, "target LNAME2")
```

The contents of field `Lname` will be copied to fields `LNAME1` and `LNAME2` in the Clustering IDT.

Append Syntax

The Append transform has the following syntax:

```
Xform (APPEND, "target field-name, order")
```

`field-name` specifies the field to which the current field will be appended. `order` specifies the order in which to append multiple fields to `field-name`. Specify a number, starting from 1.

Example:

```
FIELD=Name1, C, 45, Xform(APPEND, "target NAME,1")
FIELD=Name2, C, 45, Xform(APPEND, "target NAME,2")
```

All trailing blanks from field `Name1` will be removed and `Name2` will be appended to it and the result inserted into the field `NAME`.

Append-String Syntax

The Append transform has the following syntax:

```
Xform (APPEND-STRING, "string")
```

`string` will be appended to the source field string and copied to the target field.

Example:

```
FIELD=Surname, C, 15, Xform(APPEND-STRING, ":")
```

The character ":" will be appended to the `Surname` field.

Concatenate Syntax

The Concatenate transform has the following syntax:

```
Xform (CONCATENATE, "target field-name, order")
or
Xform (CONCAT, "target field-name, order")
```

The `field-name` specifies the field to which the current field will be concatenated. `order` specifies the order in which to concatenate multiple fields to `field-name`. Specify a number, starting from 1.

Example:

```
FIELD=Name1, C, 45, Xform(CONCATENATE, "target NAME,1")
FIELD=Name2, C, 45, Xform(CONCATENATE, "target NAME,2")
```

`Name2` will be concatenated to `Name1` and the result inserted into `NAME`.

Fill Syntax

The Fill transform has the following syntax:

```
Xform (FILL, "string")
```

The target field will be populated with the pattern defined in parameter `string`.

Example:

```
FIELD=Filler, C, 45, Xform(FILL, "+-+")
```

The field `Filler` will be filled with the string `"+-+"` repeatedly until the whole 45 characters are used.

A special expression `"Numeric(NUM)"` can be used instead of a literal string. This means that the target field will be filled with the character value of the decimal number `NUM` using the computer's native character set. For example,

```
FIELD=Tabs, C, 2, Xform(FILL, Numeric(9))
```

causes the field `Tabs` to be filled with the tab character (0x09) on computers using the ASCII character set.

Insert-Constant Syntax

The Insert-Constant transform has the following syntax:

```
Xform (INSERT-CONSTANT, "string")
```

The `string` will be inserted in the target field.

Example:

```
FIELD=Title, C, 10, Xform(INSERT-CONSTANT, "hello")
```

The `Title` field will contain the constant `"hello"`.

A special expression "`Numeric(NUM)`" can be used instead of a literal string. This means that the first position of the target field will contain the character value of the decimal number `NUM` using the computer's native character set. For example,

```
FIELD=TabAndSpaces, C, 3, Xform(INSERT-CONSTANT, Numeric(9))
```

causes the first byte of the field TabAndSpace to contain a tab character (0x09) on computers using the ASCII character set. The remaining bytes in the field - two in this example - are filled with spaces.

Insert-Field Syntax

The Insert-Field transform has the following syntax:

```
Xform (INSERT-FIELD, "target field-name, [offset]")
```

The `field-name` specifies the field to which the current field will be appended at offset. The default value of offset is 0.

Example:

```
FIELD=Addr1, C, 45, Xform(INSERT-FIELD, "target ADDR")
FIELD=Addr2, C, 45, Xform(INSERT-FIELD, "target ADDR,45")
```

The contents of `Addr1` will be copied to field `ADDR` at offset 0. The contents of field `Addr2` will be copied to field `ADDR` at offset 45.

Uppercase/Lowercase Syntax

The Uppercase/Lowercase transform has the following syntax:

```
Xform (UPPERCASE)
Xform (LOWERCASE)
```

It changes the string in the data to the specified case.

Example:

```
FIELD=Surname, C, 15, Xform(UPPERCASE)
FIELD=First,   C, 45, Xform(LOWERCASE)
```

Fields `Surname` and `First` will be converted to upper and lowercase respectively

Transformation User-Exit

A Transform User-Exit (TUE) is used to alter a record prior to insertion into the IDT.

The TUE is called after reading a record from the source file and before any field level transforms have been performed. The TUE is a "record level" exit, meaning that it has access to the entire record. Upon return from the exit, Data Clustering Engine will perform field-level transforms while converting the view record into IDT layout.

The normal use for a TUE is to insert data into a new field. Anew field can be created with the transform by specifying a name, format and length. The field can then be referenced in the TUE.

The TUE is enabled by adding a user-exit transform definition. For example,

```
VIEW-DEFINITION
*==============
NAME=DATAIN
FIELD=Name,    C, 60
FIELD=Myfield, C, 50, XFORM(user-exit, "mydll myfunc")
```

defines a field called **Myfield** which can be populated by the user-exit function `myfunc`, which is available in a shared object called `mydll`.

User-Exits must conform to a specific protocol in order to communicate with Data Clustering Engine. A User-Exit is written in C and compiled and linked as a DLL/shared library. DCE will load the User-Exit and call a predefined entry point when a "service" is required from the exit.

The parameters and other details for Transformation User-Exits are only available on request.

Cluster File

The Cluster File is a database file used to record cluster membership information created by the Data Clustering Engine. The clustering process creates at least one membership record for each Data record.

As stated earlier, the view processor will allow access to the fields in the Cluster File when they are requested via a View Definition. The following fields are available:

Field	Description
Clustering-Id, F,2	The `ClusteringId` number associated with this Cluster (membership) record.
Cluster, G,4	The cluster number associated with this Cluster record. This field along with `Clustering-Id` uniquely identifies a cluster.
Max-Score, G,1	The highest score achieved by this record.
Score-Count, I,4	The number of records that this record was scored against during the clustering process.
Score-Accepted, I,4	The number of records that had an Accepted score when processing this member.
Record-Id, G,4	The physical address of the Data record that generated this Cluster record.
Attributes, I,4	The special attributes (if any) associated with this membership record. Refer to the *Membership Attributes* section for details.

Output Views

Output views are used to format search results returned by the Search Server. Rows are returned in IDT format when no output view has been specified or in a different format when an output view has been defined.

Output Views are also used to inject search statistics into the output.

Statistical Fields

When the following field names are specified in the output view, the Search Server will provide statistics gathered during the search process.

Note that some data in the output of a search is specific to each row. For example, the score is a property that changes for each row. Other statistics are relevant to the search set as a whole. For example, the number of candidates selected is the same for each row returned because it is a property of the set.

Also note that it is not normally possible to retrieve search statistics if the search does not return any rows. In this case, a dummy IDT row (filled with asterisks) can be returned, solely as a vehicle to return the statistics. This feature is enabled with the Search Option, UNMATCHED_STATS.

Field Name	Description
IDX-IO	Number of IDX rows retrieved
IDT-IO	Number of IDT rows retrieved
IDS-MS-SEARCH-NUM	Ordinal number of the successful search within a multi-search definition (starting from 1; 0 is returned for unsuccessful searches; see Multi-Search parameter `SEARCH-LIST=` for more details)
XXX-ACCEPTED-COUNT	Number of records accepted by scoring
XXX-UNDECIDED-COUNT	Number of undecided records
XXX-REJECTED-COUNT	Number of rejected records
XXX-TOTAL-COUNT	Total number of records scored

where

XXX is `KPSL` (Key-Pre-Score-Logic), `KSL` (Key-Score-Logic), `PSL` (Pre-Score-Logic) or `SL` (Score-Logic).

USER SOURCE TABLE Section

The User Source Table (UST) section specifies how to build the IDT by sourcing data from an SQL database.

The following types of SQL data extraction are supported. An IDT can be built from:

- a single UST
- multiple USTs with a join operation
- multiple USTs with a merge (union) operation

The Loader can transform fields while extracting them from the data source. It can concatenate fields, insert text and change the case of the source fields, as described below (also refer to the *Transformations* section).

General Syntax

The UST section uses an SQL-like syntax. The following general points should be noted:

- lines beginning with two dashes "--" are treated as comments.
- tokens can be substituted with the value of an environment variable by specifying the environment variable name surrounded by #s. eg the Source Schema could be specified as #myschema#. When parsed it would be substituted with the value of the environment variable *myschema'*.
- each table definition is terminated by a semicolon.
- multiple table definitions are permitted in this section.

UST Data Types

The following tables show the data types supported for the various database types, and what they are converted to for storage in the IDT.

The first column shows the native data types that can be read from User Source Tables.

The second column shows the equivalent DCE data type. Data read from USTs are converted to a common data type to enable combining source data from multiple different database management systems.

Oracle

UST Data Type	DCE Data Type
	F
CHAR VARCHAR2 NUMBER (scale > 0) DATE [6]	C
	V
	B
	I
	G
NUMBER (scale=0) INT SMALLINT	N
	R
	X
	Z

[6] DATEs are converted to C(64) fields by default. The length may be overridden. The default installation date mask determines the date's format. This is specified either explicitly with the initialization parameter NLS_DATE_FORMAT or implicitly with the initialization parameter NLS_TERRITORY. It can also be set for a session with the ALTER SESSION command.

UDB

UST Data Type	DCE Data Type
	F
CHAR VARCHAR DATE[7] TIMESTAMP NUMBER (scale > 0)	C
	V
	B
	I
	G
NUMBER (scale=0) DECIMAL INTEGER SMALLINT BIGINT	N
	R
	X
	Z

[7]DATEs are converted to C(64) fields by default. The length may be overridden.

ODBC / Microsoft SQL Server

UST Data Type	DCE Data Type
SQL_CHAR	F
SQL_VARCHAR SQL_DATE[8] SQL_TIME SQL_TIMESTAMP SQL_TYPE_DATE SQL_TYPE_TIME SQL_TYPE_TIMESTAMP SQL_NUMERIC[9] SQL_DECIMAL[9] SQL_FLOAT SQL_REAL SQL_DOUBLE SQL_GUID	C
SQL_BINARY SQL_VARBINARY	B
SQL_NUMERIC[10] SQL_DECIMAL[10] SQL_INTEGER SQL_SMALLINT SQL_TINYINT SQL_BIGINT	N

[8]DATEs are converted to C(64) fields by default. The length may be overridden.

[9]scale > 0 (floating point numbers)

[10]scale = 0 (whole numbers)

Single UST

The simplest form of IDT is created from column values extracted from a single User Source Table. The general form of the syntax is:

```
logical-file-definition
*======================
NAME=               lf-input
PHYSICAL-FILE=      "VirtualTable"
COMMENT=            "input from SQL database"
FORMAT=             SQL
...
Section: User-Source-Tables

CREATE_IDT VirtualTable,FileNo
SOURCED_FROM [connection_string]
source_clause
[TRANSFORM transform_clause]
[SELECT_BY select_clause]
```

```
[NOSYNC]
;
```

where

VirtualTable

This is the name specified by the PHYSICAL-FILE parameter of the Logical-File-Definition.

FileNo

This is a unique file number associated with this IDT. It must be in the range 1-14999, although some `FileNo`s are reserved for internal use (2000-2010).

Connection String

The `connection_string` is used to specify connection information for access to the UST on the source database.

The format of the `connection_string` is as follows:

```
odb:99:Userid/Password@Service
```

where

Userid

DBMS user-id

Password

DBMS password

Service

This is the `Service_Name` defined in your `odbc.ini` file. Refer to the *Configuring ODBC* section for details.

For example, `odb:99:scott/tiger@ora920` specifies an Oracle host DBMS. DCE will connect to the DBMS identified as "`ora920`" using the user id "scott" and a password of "tiger". Any valid userid with SELECT privileges on the source tables may be used.

Source Clause

This section is used to specify the source of the data. The `source_clause` is used to nominate the UST columns that are to be extracted when creating the IDT. Source fields can be either added directly into the IDT or used in a transformation process, the result of which is added to the IDT.

The `source_clause` syntax is:

```
[src_schema.]table.column [tgt_fld [fmt(len)]] [,...]
```

where

src_schema

This is the name of the schema that the `table.column` belongs to. The default value is your userid.

table

The source table name that contains the column.

Oracle:

Synonyms (private, public, recursive) may be used but are converted to the real `schema.table` name during parsing.

column

The name of the source column to be extracted.

tgt_fld

The name of the IDT field. If omitted, it defaults to the value of `column`. If `tgt_fld` begins with a $, it is treated as a `virtual field`. Virtual fields are not stored in the IDT. They can only be referenced in a `transform_clause`.

fmt(len)

The format and length of the `tgt_fld` in the IDT. When omitted, they default to the format and length of the source column. Valid formats are defined in the *File Definition* section of this manual.

Transform Clause

A `transform_clause` is really an optional part of a `source_clause` or `merge_clause`. It is used to specify how virtual fields are to be combined/transformed and characteristics of the resulting field stored in the IDT.

The `transform_clause` syntax is:

```
transform_rule tgt_fld fmt(len) [order n] [,...]
```

where

transform_rule

Nominates the transformation process which will convert virtual fields into a value to be stored in `tgt_fld`.

tgt_fld

This is the name of the field to be stored in the IDT.

fmt(len)

This is the format and length of the `tgt_fld`.

order n

This is used to override the default order of fields in the IDT. Normally fields are placed in the IDT in the order of definition in the `source_clause` and `transform_clause`. You may override this order by nominating an integer value for n, starting from 1.

Transform Rules

Transform rules fall into three categories:

1. rules that use no virtual fields (`Source-Id`, `Insert-Constant`)
2. rules that operate on a single virtual field (`Upper`, `Lower`)

3. rules that operate on many fields (`Append`, `Concat`,`Insert-Field`)

Each virtual field can only be referenced once in the `transform_clause`. In the unlikely event that the same source column is to be used in more than one transform clause, add an extra `source_clause` for that column but give it a different virtual field name.

A `transform_rule` is one of the following:

```
Source-Id
Insert-Constant ("text")
Upper (vf)
Lower (vf)
Insert-Field (vf, offset, ...)
Append ( vf | lower (vf) | upper (vf) | "text" , ... )
Concat ( vf | lower (vf) | upper (vf) | "text" , ... )
```

where

Source_Id

This is a transform that generates a unique ID into `tgt_fld`. It should only be used in conjunction with the AUTO-ID option. The name of the `tgt_fld` must match that defined in `AUTO-ID-FIELD` in the *IDX-Definition* section.

Insert-Constant

This will inject a string of text into `tgt_fld`.

vf

This is the name of a virtual field defined in the `source_clause` (including the leading '$').

Upper

This will convert the virtual field `vf` to upper case.

Lower

This will convert the virtual field `vf` to lower case.

Insert-Field

This is a transform that will combine a number of virtual fields into `tgt_fld`. Each field is stored at the specified offset in `tgt_fld`. Offsets start from 0.

Append

This is a transform which will combine virtual fields and text by stripping trailing spaces from all fields and joining them together into `tgt_fld`.

Concat

This is the same as `Append` but will add a single space between the fields when joining them.

Select Clause

Is any valid SQL expression that can be used to select a *subset of records* from the UST. Data Clustering Engine does not parse this expression. It is simply added to the WHERE clause generated by DCE. Therefore it is important to ensure its correctness, otherwise a run-time SQL error will occur in the Loader.

Do not try to limit the number of rows loaded from the UST using a physical limit. For example,

```
select_by ROWNUM <= 1000
```

This approach instructs the SQL Optimizer to return the first 1000 rows but has the disadvantage that the "first 1000 rows" may not be the same ones when the Project is loaded a second time. It will produce inconsistent results.

Use a logical limit when selecting a subset of records from the UST. For example,

```
select_by EmpId >= 50000 AND EmpId <= 51000
```

This ensures a repeatable set of records.

NOSYNC Clause

DCE doesn't support data source synchronisation. Only the data that is present in the UST(s) at the time the Loader is started will end up in the clustering IDT. The NOSYNC keyword is optional.

Example

```
CREATE_IDT
        SQLinput,1
SOURCED_FROM ora:99:scott/tiger@server734
        SCOTT.EMP.EMPNO   EmpNo,
        SCOTT.EMP.ENAME   EmployeeName
SELECT_BY
        scott.emp.empno > 7800
;
```

The example will create a virtual input table named `SQLinput`. This table can then be referred to in a `logical-file-definition` as the logical input file for a clustering job:

```
logical-file-definition
*======================
NAME=           lf-input
PHYSICAL-FILE=  "SQLinput"
COMMENT=        "input from Oracle database"
FORMAT=         SQL
AUTO-ID-NAME=   JobN
```

Data will be extracted from an Oracle service named `server734` using the userid `scott` whose password is `tiger`. The IDT will contain two fields, `EmpNo` and `EmployeeName`. They will have identical formats and lengths as their corresponding source fields, `EMPNO` and `ENAME` taken from table `SCOTT.EMP`. Only employees with employee numbers greater than 7800 will be extracted and loaded into the IDT.

Joining USTs

A virtual input table may be created by joining two or more USTs from a single source database. This would normally be done if the source tables were normalized into multiple tables but the IDT requires data from all source tables. The syntax is identical to the Single USTs syntax with the addition of a `join_expression`:

```
CREATE_IDT VirtualName,FileNo
       SOURCED_FROM connection_string
             source_clause
       [TRANSFORM transform_clause]
       JOIN_BY join_expression
       [SELECT_BY select_expression]
       [NOSYNC]
       ;
```

Source Clause

Refer to the *Source Clause* description in the *Single UST* section.

Transform Clause

Refer to the *Transform Clause* description in the *Single UST* section.

Join Expression

The *primary table* is the first UST table mentioned in the `SOURCED_FROM` clause. A secondary table is a UST joined to the primary table using a foreign key stored in the primary.

A `join_expression` is used to specify how to join the primary table to one of the secondary tables or more generally, how to join a parent table to a child.

A `join_expression` must be provided for each pair of tables to be joined. It defines the relationship between tables, where the parent contains the foreign key of the child.

It is of the form,

```
parent_table_column = child_table_column [AND p_col = c_col] ,...
```

where,

parent_table_column

This is a fully qualified column name in the *parent table*.

child_table_column

This is a fully qualified column name in the *child table*.

p_col

This is an unqualified column name in the *parent table* (for compound keys).

c_col

This is an unqualified column name in the *child table* (for compound keys).

All **parent_table_columns** specified in a `join_expression` must be included in the created virtual table as non-virtual fields.

An outer join is performed on the *primary table*, meaning that all rows in the primary table will be added to the virtual table even if they failed to be joined to any secondary tables. In this case, the columns extracted from the secondary table are set to NULL.

Example

The virtual input table `SQLinput` is to be created. Columns are extracted from two tables, `NAMEADDR` and `OTHERDETAILS`. These tables belong to schema `#SSA_UID#` which is evaluated at parse time using the environment variable *SSA_UID*. The tables are joined using the `EMPNO` column in `NAMEADDR` and the `SSN` column in table `OTHERDETAILS`.

Various transformations are used. Columns given and family are concatenated to form a field called `NAME`. The order parameter is used to change the default ordering of fields in the IDT to: `myid`, `NAME`, `EMPNO`, `SSN`, `TITLE`, `Addr` and `Phone`.

```
CREATE_IDT
        SQLinput,50
SOURCED_FROM ora:99:scott/tiger@server734
        #SSA_UID#.NAMEADDR.EMPNO            EmpNo N(25),
        #SSA_UID#.NAMEADDR.given            $given,
        #SSA_UID#.NAMEADDR.family           $family,
        #SSA_UID#.NAMEADDR.ADDR             $addr,
        #SSA_UID#.OTHERDETAILS.SSN,
        #SSA_UID#.OTHERDETAILS.PHONE        $phone
TRANSFORM
        source-id                           myid   f(10) order 1,
        insert-constant("hello there")      title  c(15),
        upper($addr)                        Addr   c(30),
        lower($phone)                       Phone  c(12),
        concat($given,$family)              NAME   c(50) order 2
JOIN_BY
        #SSA_UID#.NAMEADDR.EMPNO = #SSA_UID#.OTHERDETAILS.SSN
```

```
NOSYNC
;
```

Merging USTs

A virtual input table file can be created by merging the contents of two or more User Source Tables. Multiple heterogeneous source databases are permitted. The columns extracted from the tables are mapped into a common format using multiple `merge_clause` and `transform_clause` pairs (one pair per UST).

```
CREATE_IDT VirtualTable,FileNo
       MERGED_FROM [connection_string] merge_clause
       [TRANSFORM transform_clause] ...
[NOSYNC]
;
```

Source Clause

Refer to the *Source Clause* description in the *Single UST* section.

Transform Clause

Refer to the *Transform Clause* description in the *Single UST* section.

Merge Clause

The `merge_clause` is identical to the `source_clause` in syntax but its semantics differ:

- The first `merge_clause/transform_clause` pair is used to define the virtual table column names, formats, lengths and order.
- The second and subsequent pairs define the mapping from source columns in other USTs to the `tgt_flds` defined in the first pair. They can not specify format, length nor order. The `tgt_fld` names must match those defined by the first pair.

Example

The virtual input table `SQLinput` is to be created from data extracted from tables `EXEMPL` and `EMPLOYEES`. These tables are found on different databases (mars and jupiter respectively).

The common layout of the virtual table is defined by the first `merge_clause / transform_clause` pair as:

```
MyId   C(10)
NAME   C(50)
Addr   C(50)
SSN    format/length same as EXEMPL.SSN
```

Columns `ADDR_L1`, `ADDR_L2`, `ZIP` from table `EXEMPL` will be concatenated to form a value for Addr.

Columns `GIVEN` and `FAMILY` from table `EMPLOYEES` will be concatenated to form a value for `NAME`. The field `EMPNO` in `EMPLOYEES` maps to `SSN`.

```
CREATE_IDT
        SQLinput,51
MERGED_FROMora:99:scott/tiger@mars
        #SSA_UID#.EXEMPL.FULL_NAME     NAME C(50),
        #SSA_UID#.EXEMPL.SSN SSN,
        #SSA_UID#.EXEMPL.ADDR_L1       $a1,
        #SSA_UID#.EXEMPL.ADDR_L2       $a2,
        #SSA_UID#.EXEMPL.ZIP           $zipcode
TRANSFORM
        source-id                      MyId c(10) order 1,
        concat($a1,$a2,$zipcode)       ADDR c(50) order 3

MERGED_FROMora:99:buzz/lightyear@jupiter
```

```
        #SSA_UID#.EMPLOYEES.EMPNO      SSN,
        #SSA_UID#.EMPLOYEES.GIVEN      $given,
        #SSA_UID#.EMPLOYEES.FAMILY     $family,
        #SSA_UID#.EMPLOYEES.ADDR       ADDR
TRANSFORM
        concat($given,$family)         NAME;
```

Defining Source Tables

The UST Section is also used to define database source tables to be used to supply input records to a Relate batch search process. The syntax is similar to the `CREATE_IDT` clause, with a few minor differences:

- `DEFINE_SOURCE` replaces `CREATE_IDT`,
- File-Ids are not required

The syntax is:

```
DEFINE_SOURCE SrcName
SOURCED_FROM [connection_string] source_clause
[TRANSFORM transform_clause]
[JOIN_BY   join_expression]
[SELECT_BY select_clause]
[NOSYNC]
;
```

where

SrcName

This is the name of the data source.

When the Project is loaded, a File and View named `SrcName` are created.

The View is used as an input view for Relate to describe the input records. The target field names in the `source_clause` (and therefore View) must match the field names in the IDT in order for these fields to be mapped to IDT fields.

Example

```
define_source SRC05
sourced_from ora:99:uid/pwd@svc
        #SSA_UID#.TESTX05A.RECNUM     RecNum     C(5),
        #SSA_UID#.TESTX05A.LASTNAME   $last,
        #SSA_UID#.TESTX05A.FIRSTNAME  $first,
        #SSA_UID#.TESTX05A.MIDDLENAME $middle,
        #SSA_UID#.TESTX05A.ADDR1      $a1,
        #SSA_UID#.TESTX05A.ADDR2      $a2,
transform
        concat ($last,$first,$middle) Name       C(50),
        concat ($a1, $a2)             Addr       C(40)
;
```

The following `View-Definition` is generated automatically to define the record layout for relate:

```
VIEW-DEFINITION
NAME=src05
FIELD=RecNum,C,5
FIELD=Name,C,50
FIELD=Addr,C,40
```

Flattening IDTs

Flattening is the process of packing denormalized data created by joining tables in a "one to many" (1:M) relationship, into repeating groups in the IDT. It can significantly increase search performance.

The problem

A typical database design consists of multiple tables that have been normalized to some degree. While full normalization is logically elegant and efficient to update, it performs poorly for read access when the tables need to be joined to satisfy a query.

DCE provides very fast search performance by storing all search, matching and display data together in the same table, thereby avoiding the need for costly joins at search time. The DCE Loader denormalized the data while creating the IDT.

A disadvantage of denormalization is the explosion in the number of rows that occurs when joining tables that have a 1:M relationship, as the SQL engine produces one row for every possible combination.

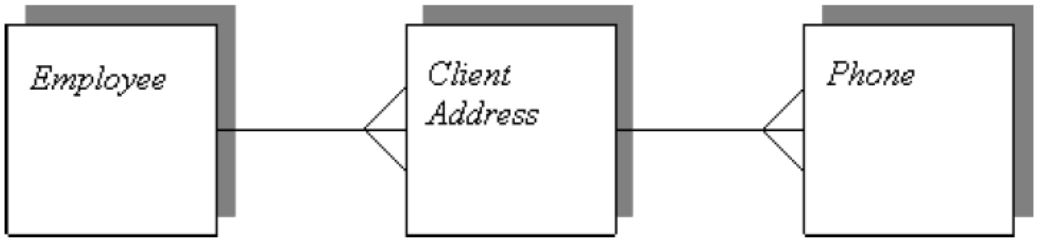
As a result, storage and retrieval costs increase.

To overcome this problem, DCE can collapse (flatten) related denormalized rows into one IDT row that contains repeating groups. This means:

- significantly faster search performance
- less database storage for the IDT and IDXs

The most significant performance benefits occur when the Key-Field for the IDX is sourced from the 1 table in the 1:M relationship. This is due to the fact that the use of repeating groups for the M data reduces the number of rows containing the same Key-Field value, which in turn produces less keys to index (at load time) and less candidates to retrieve and match at search time.

Logical Design



Consider the following logical design where the tables have been fully normalized. Each employee of a contracting business can work for many clients. Each client's business premises may have many phone lines.

Suppose there is only one employee currently:

Emp_Id	Name
E001	Joe Programmer

Joe contracts his services to three companies located around the state:

Company_Id	Address	FK_Emp_Id
C001	3 Lonsdale	St E001
C002	19 Torrens	St E001
C003	4 Rudd	St E001

Each company has the following phone numbers:

Company_Id	Phone
C001	62575400
C001	62575401
C002	98940000
C003	52985500
C003	52985501
C003	52985502

Denormalized Data

A simple SQL query to denormalize this information will generate six rows of data because Joe Programmer works at three offices, each having multiple phone numbers.

Table 1. The denormalized data from a join operation.

Emp_Id	Name	Company_Id	Address	Phone
E001	Joe Programmer	C001	3 Lonsdale St	62575400
E001	Joe Programmer	C001	3 Lonsdale St	62575401
E001	Joe Programmer	C002	19 Torrens St	98940000
E001	Joe Programmer	C003	4 Rudd St	52985500
E001	Joe Programmer	C003	4 Rudd St	52985501
E001	Joe Programmer	C003	4 Rudd St	52985502

If the search application needs to search on Name, DCE will create fuzzy keys on the Name column. As there are many rows with the same value, duplicate keys will be created and maintained. At search time, multiple rows will be retrieved and matched.

Flattened Data

To reduce the number of rows, DCE can flatten the six rows into one. This is achieved by declaring some columns in the IDT as repeating fields . If data sourced from the M tables (`Company_Id`, `Address` and `Phone`) were defined to repeat up to six times, all of the data could be flattened into one row.

Note: The table below has been turned on its side due to space limitations. It represents a single row of data with column names on the left and data values on the right. Note how the data from the "1" Table no longer repeats.

Table 2. A single flattened row

Column	Value
Emp_Id	E001
Name	Joe Programmer
Company_Id [1]	C001
Company_Id [2]	C002
Company_Id [3]	C003
Address [1]	3 Lonsdale St
Address [2]	3 Lonsdale St
Address [3]	19 Torrens St
Address [4]	4 Rudd St
Address [5]	4 Rudd St
Address [6]	4 Rudd St
Phone [1]	62575400
Phone [2]	62575401
Phone [3]	98940000
Phone [4]	52985500
Phone [5]	52985501
Phone [6]	52985502

An IDT with this structure would have only one row and the IDX would contain six times fewer `Name` keys. The number of candidates selected during a search on the `Name` IDX would also decrease by a factor of six.

Note: This structure does not improve the performance of an IDX created on the `Address` fields, as they contain duplicate values. However, Flattening-Options (discussed later) can be defined to remove duplicate entries from the repeating fields in order to provide some benefit as well.

Syntax

Flattening is enabled by defining

- the layout of the denormalized (joined) data,
- the layout of the IDT including the maximum number of occurrences for repeating fields,
- the columns used to determine how to group denormalized data (Flattening-Keys), and
- Flattening-Options.

Note: When flattening is enabled, the input must be read and processed using `LOADIT`. Do not schedule any utilities that read the input more than once. For example, using a `PRE` or `SORTIT` step will run flattening multiple times, producing incorrect results.

IDT-Definition

The Denormalized-View keyword is used to define the name of the View that provides the layout of the denormalized data. The denormalized data can be read from either a flat-file or UST. If read from a flat-file, you must provide a `View-Definition`. If the data is sourced from USTs, a `View-Definition` is generated automatically from the `User-Source-Tables` section. The generated view's name is the IDT name, suffixed with "-DENORM".

The Flatten-Keys keyword is used to define the columns used to group denormalized rows. The denormalized data is sorted by the `Flatten-Keys` and all rows with the same key value are packed into the same flattened row. Refer to the *Flattening Process* section for details.

The Options keyword in the `IDT-Definition` is used to specify various flattening options that affect how data is packed into the repeating groups. Refer to the *Flattening Options* section for details.

For example,

```
IDT-Definition
*=============
NAME=               IDT112
DENORMALIZED-VIEW= IDT112-DENORM
FLATTEN-KEYS=      "EmpId"
OPTIONS=           Flat-Remove-Identical
```

IDT Layout

The IDT layout can be provided as a File-Definition (when sourcing from a flat-file) or from the User- Source-Table section when sourcing from USTs. The IDT layout must be identical to the Denormalized- View (same column names, types, and order) with the exception that some columns are defined to repeat.

Repeating fields are defined by immediately following the target-field name with [`n`] where `n` represents the maximum number of occurrences for a repeating field.

For example,

```
Section: User-Source-Tables

Create_Idt
        IDT112
sourced_from odb:99:ssa/ssa@ora817
        Employee.EmpId       EmpId,
        Employee.Name        Name,
        Address.CompanyId    CompanyId [6],
        Address.Address      Address [6],
        Phone.Num            Phone [6]
join_by
        Employee.EmpId    = Address.EmpFK,
        Address.CompanyId = Phone.CompanyId
;
```

Flattening Process

Denormalized rows are read using the Denormalized-View and then sorted by the flattening keys.

The sorted data is used to build IDT rows. Data is moved from denormalized rows into an IDT row. An IDT row is written out either

- at the break of a flattening key value, or
- when the IDT row is full.

The latter will occur when a repeating field is full. If more data arrives with the same key value, additional IDT rows are built with the same key value.

The construction phase also verifies that non-repeating fields have the same value in all denormalized rows because it is not possible to store more than one value. An incorrect design or selection of the flattening keys can result in this situation. If this occurs the load will issue warnings similar to this:

```
Warning: Illegal break in denormalized record n, field f: 'xxx'
```

Flattening Options

Options for controlling how data is packed into repeating fields can be specified with the IDTDefinition's OPTION= parameter:

Flat-Keep-Blanks

By default, blank fields are not moved into a repeating group. This option keeps blank fields, thereby maintaining positionality within the repeating fields. This makes it possible to relate data from two repeating fields. For example, the `nth Company_Id` is related to the `nth Address`.

Flat-Remove-Identical

By default, identical values in a repeating field are kept, making it possible to relate data from two repeating fields. It requires the same number of repeats for all repeating groups.

With this option enabled duplicate values are removed, so each repeating group can be sized according to the number of unique values it is expected to hold. This option does not maintain positionality.

Removal of identical values only applies to the current IDT row under construction. If a repeating field overflows causing another IDT row to be created with the same key values, the values stored in the previous row are not considered when searching for duplicates.

Tuning / Load Statistics

The Loader produces statistics that can be used to select appropriate values for repeat counts. For example:

```
Flatten:.............................Denormalized-In    4196
Flatten:                            Unique-Keys        2094
Flatten:                            IDT Out            2894
Flatten:                              alt_ent_num           [ 2]
Flatten:............................   [  0]      1585    75.69%
Flatten:                              [  1]       363    93.03%
Flatten:                              [  2]        71    96.42%
Flatten:                              [  3]        24    97.56%
Flatten:............................   [  4]        14    98.23%
Flatten:                              [  5]         9    98.66%
Flatten:                              [  6]         9    99.09%
Flatten:                              [  7]         3    99.24%
Flatten:............................   [  8]         2    99.33%
Flatten:                              [  9]         7    99.67%
Flatten:                              [ 10]         3    99.81%
Flatten:                              [ 11]         0    99.81%
```

```
Flatten:.............................   [ 12]         0    99.81%
Flatten:                               [ 13]         0    99.81%
Flatten:                               [ 14]         0    99.81%
Flatten:                               [ 15]         2    99.90%
Flatten:.............................   [ 16]         0    99.90%
Flatten:                               [ 17]         0    99.90%
Flatten:                               [ 18]         1    99.95%
Flatten:                               [ 19]         0    99.95%
Flatten:.............................   [ 20]         0    99.95%
Flatten:                               [ 21]         0    99.95%
Flatten:                               [ 22]         0    99.95%
Flatten:                               [ 23]         0    99.95%
Flatten:.............................   [ 24]         0    99.95%
Flatten:                               [ 25]         0    99.95%
Flatten:                               [ 26]         0    99.95%
Flatten:                               [ 27]         1   100.00%
```

This report shows that 4196 rows were created as a result of the denormalization (SQL join) process, while 2094 of those rows contained unique keys (`FLATTEN_KEYS=`). After flattening, 2894 rows were loaded to the IDT. This means that 800 rows overflowed to secondary rows because the number of repeats was insufficient to hold all occurrences.

The next part of the report shows a histogram for each repeating field. The field `alt_ent_num` was defined to have two repeats ([2]) for this load test. The histogram tabulates the actual number of repeats in each normalized row (prior to flattening) and a cumulative percentage of rows. For example, the line

```
Flatten:                           [ 1]        363    93.03%
```

states that 363 rows has only one occurrence, and this accounts for 93% of the total rows. The table also tells us that the maximum number of occurrences was 27, and that 1585 rows had no value for this field.

The cumulative percentage can be used to select an appropriate value for the number of repeats. For example, if `alt_ent_num` were defined to have six repeats, over 99% of values would fit into one IDT row (no overflows).

Maintaining the Project

The Project Definition File is initially created with a text editor and loaded into the Clustering Rulebase when the Project is created.

After the Project Create step, it is possible to edit the Project using the Edit option under the Project heading in the Console window. This will launch the Project Editor where any amendments can be made to the selected Project. See the *Editing a Project* section for more details.

Once you are done with your changes you can either select **Save**, which will just save your changes but not actually load the Project, or select **Save & Load**, which will perform the Project load task committing your changes to the Project. You can then reinitialize the database and run the Clustering tasks with the amendments you have just made.

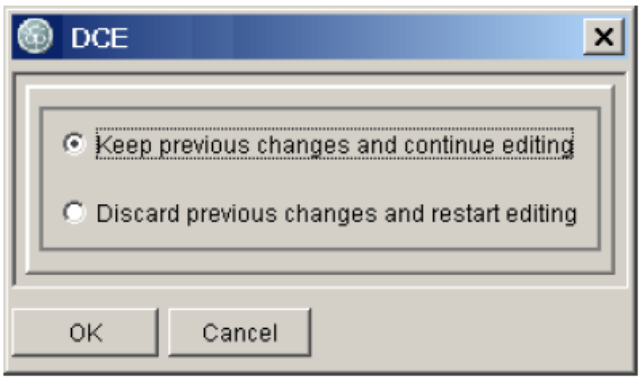
Restarting or Continuing an Edit Session

Editing a Project works by making a copy of the Project, allowing the user to modify the copy, saving the copy and, finally, loading the Project, which involves replacing the Project with the modified copy. This mechanism allows a partially edited Project to be saved without any affect on the original Project.

Only when the **Save & Load** option is selected, an attempt is made to commit the changes to the Project. If the Load fails for any reason, the original Project is restored to its original, pre-Load condition. The edited

copy of the Project is kept so that the user can re-edit the Project, correct the error and try the Load process again.

- If **Save** was selected at the end of the last Edit Session, or if an error occurred when **Save & Load** was last selected, then the next time the **Project** > **Edit** option is selected the following dialog will be displayed:



- Choose the Continue editing option to carry on editing where you left off previously. Or choose **Restart** editing to discard your earlier changes. Choose this option with care.

  Note: After **Save & Load**, before re-running Clustering tasks, you must re-initialize the database to remove records from previous runs. Any alterations, which you make to the Project using the Project editor, will NOT be reflected back to your `.sdf` text file. In other words, this action only affects the Projects currently contained in the Rulebase.

Editing a Project

This section provides information how to edit a Project.

Project Editor

The Project Editor provides an easy to use interface to change the rules that define a Project. It is sensitive to the state of objects and prevents editing an object's definition when it has already been implemented. For example the layout of an IDT cannot be changed if it has been loaded. Similarly, you cannot change key building rules for an IDX if it is already loaded.

Note: The Project Editor can only be used to edit Projects created using the Console. A Project loaded using low-level batch utilities cannot be edited.

Starting

The Project Editor is invoked from the Console Client's **Project** > **Edit** button. The Editor works on a *copy of the rules* for a given Project. If you have ended a previous editing session without Loading the rules, or the Load failed, the Editor will prompt you to either:

- Keep previous changes and continue editing, or
- Discard previous changes and restart editing

Editing

A tree structure is used to navigate to the object you wish to edit. Objects that belong in a logical hierarchy will appear in the tree structure in two places:

- as an individual node attached to the root of the tree, and
- as part of a hierarchy.

For example, IDXs are attached to their parent IDT and also to the root of the tree. The individual nodes attached to the root may be edited, cloned, etc. The nodes in the hierarchical tree structure are present only to highlight the relationship between objects and can not be modified.

If you make any changes to an object you must either **Save Changes** or **Discard Changes** before moving to another object. Saving and discarding only effects the copy of the rules.

The copy of the rules is not parsed until you click the **Load** button. A successful Load process will replace the existing rules with the copy and deletes the copy of the rules. If the Load process fails, the original Project remains in effect and the copy of the rules you have been editing is available for correction by restarting the Editor.

The **Close** button will exit the editor without saving the most recent changes you have made.

Cloning and Adding

An existing object can be cloned to create a new object of the same type. You will be prompted to name the new object after which you can edit it.

If you wish to add a new object that does not already exist, you will need to use the **Add** button. Click the Project name in the tree structure and then click the **Add** button. Select the type of object you wish to add and enter a name for it.

Help

Help is available as tool tips. Hover the mouse pointer over the item that you want help for and a tool tip appears.

Advanced Options

By default, the Editor will only show the most common options that the average user will want to see. If you wish to use some of the more advanced options, click the **Show Advanced** button to enable access to all options for a given object.

Project Template

The GUI Project Editor uses a project template to add new components (IDTs, IDXs, Searches, etc). A project template is provided in the (DCE Server) installation in `dce/projects/project_template.pr`. A new template is shipped with every Fix CD. The console performs verification of the template when the console is started, when a new Rulebase is created or when a different Rulebase is selected. If the current template is not up to date then the template will be imported from the new template file. The template must not be edited.

Backing Up the Database and Index

For performance reasons the Clustering database does not have any restart / recovery features as would be found on many commercial database systems. In the event of a power failure or disk crash during a long clustering task the database and index will most likely be corrupted and the task would need to begin from the start again.

To avoid such a situation it is possible to create a backup of the necessary files, but the `CLUSTER` step needs to be in the wait state (refer to the *Stopping and Restarting Clustering* section) for you to do this. Once in a wait state, close the Console Client, bring down the DCE Server and create a backup copy of the following files from the Server Working Directory:

- *.db
- *.ndx

If a problem occurs, then these files can be restored to their original location and the Resume Clustering feature of the Console used to start again from the time that the backup was taken.

Note: Some environment variables such as the work directory name are expanded and file names are stored in the Rulebase in expanded form. This means that the backed up files must be restored to the original location.

Membership Attributes

The clustering process creates Cluster membership records. Each record contains status information about the "membership" of a data record in a particular cluster.

The layout of the fields used to store membership information is documented in the *Customized Format Definition Syntax* section.

A member in a cluster is assigned certain attributes that determine how it behaves during the clustering process:

- Voting / Non-Voting
- Undecided

Attributes are set when a membership record is created. Attributes are used while determining new members for a cluster.

Some of the Attributes are also present in the indexes. This allows efficient processing of the database while selecting records for a nominated set of membership attributes. This subset is referred to as the Selection Attributes.

Voting Attribute

A cluster member can be either Voting or Non-Voting. If it has voting rights, it can be used in the scoring process when new records are being matched to the cluster it participates in. Non-voting members do not participate in the scoring process and therefore can not attract new members to the cluster. The following Job Options are used to control the Voting rules.

Set Attributes

- `SET-ALL-VOTE` - all members, which are added, can vote
- `SET-NONE-VOTE` - all members, which are added, can not vote

- SET-HEADERS-VOTE- only the founding member of the cluster can vote (one per cluster)

Use Attributes

USE-ATTRIBUTES honour the voting attribute

Defaults

The default is SET-ALL-VOTE, --USE-ATTRIBUTES meaning do not honour the attributes and set all new members as Voting. To enable the use of attributes, specify USE-ATTRIBUTES.

When running CLUSTERING-METHOD=MANY without specifying the job option NO-ADD (ie. we are adding new records), some restrictions are enforced: SET-ALL-VOTE can not be specified and neither can --USE-ATTRIBUTES. If you explicitly specify these parameters an error message will be produced. It is assumed that you have seeded or preclustered a database and then want to run the MANY clustering to add new records.

Post

You may select output records for the POST phase based on the Attributes by specifying the following Output-Options:

VOTING means report only Voting records.

--VOTING means report only Non-Voting records.

Undecided Attribute

When a record is found to score below the Accept limit but not below the Reject limit then it is added to the Cluster as a Non-Voting member. It is also marked with the Undecided Attribute. When processing Non-Voting (or all records), records with this Attribute will be retrieved.

This Attribute can be used to determine how a record came to be added as a Non-Voting member:

- an undecided record has the Undecided attribute,

whereas,

- records added with a Set-None-Vote option, or
- a non-header record added with the Set-Headers-Vote option

do not have this attribute.

Performance Optimization

The Clustering Process reads records from the database or input file. For each record, it generates a search range using the KEY-FIELD. The Name Index is searched to create a list of candidate records with similar keys. The set of candidates are then read from the database and scored against the search record.

The process can be optimized by

- reducing the size of the candidate set, thereby reducing the amount of scoring required, and/or
- reducing the cost of scoring two records
- utilizing multiple CPUs
- reducing database I/O

The following sections discuss ways in which to achieve these goals.

Partitions

This approach is used to reduce the size of the candidate set.

For very large files, the key generated from the `KEY-FIELD` may have a high selectivity due to the sheer volume of data present on the file. Therefore searching for candidates using the key will create very large candidate sets.

If the nature of the data is well understood, it may be possible to qualify the key with additional data from the record so that the "qualified key" becomes more selective.

The PARTITION option instructs the Data Clustering Engine to build a concatenated key from the `KEY-FIELD` and up to five fields/sub-fields taken from the data record. The partition information forms a high-order qualifier for the key (it is prefixed to the key).

For example, an application may wish to cluster all names in a telephone directory. If we are willing to only examine candidates from a particular region, we could partition the data using a post-code or some other information that can group the candidates into regions. Performance is improved by reducing the size of candidate sets. The disadvantage is that candidates will only be selected from within regions; not outside. If this makes sense from the perspective of the "business problem" being solved then partitioning can be used.

The Data Clustering Engine has an option to write statistics to the database log. If this option is set and you choose to cluster using a partition with many values, statistics will be written for each partition.

This overhead can become prohibitive for very many partitions. In this case, database logging should be turned off, otherwise the cost of logging might undo the gains made by partitioning. Refer to the *Verbosity Options* section.

Key Data / Key-Score-Logic

You can use this approach to reduce the size of the candidate set.

Use `KEY-DATA` to append redundant information from the Data record to the key generated from `KEY-FIELD`. It is stored in the index. The additional information appended to the key is passed to a Score-Logic routine, which can then decide whether or not this particular record should be included in the candidate set.

Up to five fields from the Data record can be appended to the key. Refer to the *IDX-Definition's KEY-DATA* parameter for the syntax.

A Score-Logic routine, which examines the `KEY-DATA` from the search and file records, can be provided in one of these forms:

- a standard SSA-NAME3 matching logic enabled using `KEY-PRE-SCORE-LOGIC=`
- a standard SSA-NAME3 matching logic enabled using `KEY-SCORE-LOGIC=`

See also the `FULL-KEY-DATA` under *IDX-Definition's | OPTIONS=* section.

Note: Scoring with KEY-DATA can be much more efficient than scoring with data from the Data Record (as used by Pre-Score-Logic and Score-Logic).

Key-Pre-Score-Logic / Key-Score-Logic

The SSA-NAME3 matching logic is nominated using the Clustering-Definition's KEY-PRE-SCORE-LOGIC or KEY-SCORE-LOGIC keywords.

The matching routine is called with the search and file records `KEY-DATA` fields. If the returned score is less than the limit defined for the scoring logic, the file record is rejected from the candidate set.

For example, to append the first two bytes of a postal-code field to the key and then score it using a SSA-NAME3 matching scheme, perform the following tasks:

1. Code the following definitions in `Clustering-Definition`:

```
KEY-SCORE-LOGIC=SSA,
System(default),
Population(test),
Controls ("PURPOSE=Fields MATCH_LEVEL=Conservative"),
Matching-Fields("post-code:Postal_Area")
```

2. Code the following definition in `IDX-Definition`:

```
KEY-DATA      = (POST-CODE,2,0)
```

Pre-Scoring

This approach is used to reduce the cost of scoring two records.

The most expensive part of the matching process is the Scoring. Once a set of candidate records is chosen, each candidate is scored against the search record. In practice, the scoring is often complex in nature and involves several methods scoring several fields on the search and file records. If it is possible to quickly reject a candidate record by scoring it with a "light weight" (inexpensive) scoring scheme, we can avoid the need to call a more complex scheme.

The Clustering's `PRE-SCORE-LOGIC` is used to define an optional "light weight" scoring scheme which is processed before the normal `SCORE-LOGIC`. If it returns a "no match" condition, the more expensive `SCORE-LOGIC` is not called.

Note that it is possible to apply this two stage scoring approach to Key Data by using the `Key-Pre-Score-Logic` and `Key-Logic` parameters.

Scoring Phases

The previous sections discuss the different phases at which records may be scored.

This section brings them together and discusses the use of the `Accept` and `Reject` limits. `Accept` and `Reject` limits are defined by SSA-NAME3 Population Rules and can be adjusted using the CONTROLS parameter in the Project definition.

Scoring happens in four distinct phases known as Key Pre Scoring, Key Scoring, Pre Scoring and Scoring respectively. There are three possible outcomes as a result of the score in each phase. A record may be

- rejected,
- accepted,
- passed to the next scoring phase.

Rejection occurs when the score is less than the `Reject` limit for that phase. A rejected record is not passed to any other phases. It is simply discarded.

A record is accepted when its score is greater than or equal to the `Accept` limit for that phase. A record that is accepted is assigned the score from the accepting phase. It does not participate in any further scoring phases.

A record that has a score which is `>= Reject` limit and `< Accept` limit is passed to the next scoring phase. If no more phases exist then the record is considered undecided . It is then added to the cluster but with a non-voting status.

Multiple Accept Phases

Although uncommon, some Projects use this mechanism to accept records early, rather then reject them. In the case where we have multiple accept phases and we use CLUSTERING-METHOD=BEST the DCE behaves in the following fashion: As it is possible for records to be accepted by multiple scoring phases, each with a different scheme and/or options and weights, the scores from the different phases can not be directly compared (when we wish to rank the records).

To overcome the problem of comparing apples to oranges, the DCE assigns a score of 100 to a record that has been accepted by an early phase. It assigns a score of 0 if the record is rejected by an early phase. Any records that are passed to the next phase are assigned the score returned by that phase.

Adjusting the Accept and Reject limits

`Accept` and `Reject` limits are defined by SSA-NAME3 V2 Population Rules. The standard Populations usually define an `Accept` limit to be less than or equal to 100. If a record reaches a score that is greater than or equal to the `Accept` limit for that phase, the record is accepted and the record does not take part any further scoring phases as explained above. However, if the reason for using multiple scoring phases is only to reject early and never to accept until the final scoring phase, then the default `Accept` limit is not useful. In order to never accept records in a scoring phase, one should specify a scoring phase with the `Accept` limit of 101, which can be never reached. This can be done using the following syntax in the CONTROLS parameter in the Scoring phase definition:

Where AdjA is the `Accept` level adjustment and AdjR is the `Reject` level adjustment. To force the `Accept` limit to become 101 the exact value of +101 must be used as in the following example:

where the special value +101 is forcing the `Accept` level to become exactly 101. The Reject adjustment should also be specified. Omitting the `Reject` limit adjustment means that the `Reject` level is adjusted by the same amount as the `Accept` level and the value 101 would cause all records to be rejected, which is clearly not useful. By specifying the value +0 the `Reject` limit is not changed from the original value specified in the Population Rules.

See the *SSA-NAME3 POPULATIONS and CONTROLS* manual for further details.

Utilizing multiple CPUs

If you run the Data Clustering Engine on a multi-processor computer, the number of processors is automatically detected and used as the default value for the number of key-scoring threads created. If you need to alter this behaviour, you can to do one of the following:

Note: The multi-threaded clustering feature is only available with Key-Scoring.

1. Set the environment variable *SSA_DCETHREADS=#*, where `#` is the number of threads. This environment variable should be set before the DCE Server is started or optionally you can set this variable using the Console's **Settings** > **Environment Settings** dialog.
2. Add the THREADS(#) keyword in the clustering job options in the Project definition file. The `#` indicates the number of threads that you want to make available to the key-scoring process. e.g.

   ```
   job-definition
   NAME=job-cluster
   TYPE=cluster
   OPTIONS=Threads(4)
   ```

   Note: The environment variable *SSA_DCETHREADS* setting overrides the THREADS(#) keyword. If neither is specified, DCE automatically detects the number of CPUs available.

   It is recommended that the number of key-scoring threads does not exceed the number of CPUs on your system.

Reducing Database I/O

This section provides information on reducing database input and output operation.

IDX Size

The physical size of the IDX will determine how efficiently the database cache will operate. Reducing the size of the IDXs will improve performance. This is achieved by selecting the most appropriate Compressed-Key-Data value as described in the *Compressed Key Data* section below and using flattening to reduce the number of rows (Refer to the *Flattening IDTs* section for more information).

Compressed Key Data

The IDX stores fuzzy keys and identity data for matching. The identity data is compressed and stored using an algorithm selected with the IDX-definition's Compress-Method parameter. All methods will compress the identity data and store it in the IDX together with its fuzzy key.

If the length of the IDX record exceeds the DBMS's `maximum segment length` (256 bytes) DCE can either,

- `Method 0` split the IDX record into multiple adjacent segments (which are all shorter than the length limit).
- `Method 1` truncate the IDX record at the length limit and only store one segment. This forces additional I/O at run time if the IDX record is selected for matching, as the matching data must be read from the IDT record.

IDX segments are fixed in length and have the following layout:

Partitions Fuzzy	SSA-NAME3 Key	Compressed Identity Data

The `segment length` is the sum of

1. partition length (optional, user defined)
2. SSA key length (5 or 8 bytes)
3. Compress-Key-Data(n) parameter
4. 4 bytes of additional overhead

Since the segment length is fixed, choosing an appropriate value for `n` is important because it affects the total amount of space used and the I/O performance of the index. Determining an optimal value for `n` requires knowledge of the characteristics of the source data and how well it can be compressed. If `n` is set too high, all segments will use more space than necessary. If `n` is too low, records will be split into multiple segments, incurring extra overhead for the duplication of the Partition and Fuzzy Key in each segment.

Measuring Compression

The Table Loader can be used sample the source data and produce a histogram of the Compressed Identity Data lengths. A sample table appears below:

```
loadit>    Histogram:                              KG KeyLen Count Percent
loadit>    Histogram-KG:.........................      78   2    0.21%
loadit>    Histogram-KG:                              81   4    0.64%
loadit>    Histogram-KG:                              83   1    0.75%
loadit>    Histogram-KG:                              85   6    1.39%
loadit>    Histogram-KG:.........................      86   1    1.49%
loadit>    Histogram-KG:                              87   4    1.92%
loadit>    Histogram-KG:                              88   3    2.24%
loadit>    Histogram-KG:                              89  14    3.73%
loadit>    Histogram-KG:.........................      90  10    4.80%
loadit>    Histogram-KG:                              91  16    6.50%
loadit>    Histogram-KG:                              92   9    7.46%
loadit>    Histogram-KG:                              93  17    9.28%
loadit>    Histogram-KG:.........................      94  19   11.30%
```

```
loadit>    Histogram-KG:                                  95  15   12.90%
loadit>    Histogram-KG:                                  96  32   16.31%
loadit>    Histogram-KG:                                  97  31   19.62%
loadit>    Histogram-KG:.........................         98  29   22.71%
loadit>    Histogram-KG:                                  99  40   26.97%
loadit>    Histogram-KG:                                 100  45   31.77%
loadit>    Histogram-KG:                                 101  25   34.43%
loadit>    Histogram-KG:.........................        102  31   37.74%
loadit>    Histogram-KG:                                 103  28   40.72%
loadit>    Histogram-KG:                                 104  34   44.35%
loadit>    Histogram-KG:                                 105  46   49.25%
loadit>    Histogram-KG:.........................        106  30   52.45%
loadit>    Histogram-KG:                                 107  34   56.08%
loadit>    Histogram-KG:                                 108  40   60.34%
loadit>    Histogram-KG:                                 109  21   62.58%
loadit>    Histogram-KG:.........................        110  34   66.20%
loadit>    Histogram-KG:                                 111  37   70.15%
loadit>    Histogram-KG:                                 112  48   75.27%
loadit>    Histogram-KG:                                 113  27   78.14%
loadit>    Histogram-KG:.........................        114  28   81.13%
loadit>    Histogram-KG:                                 115  21   83.37%
loadit>    Histogram-KG:                                 116  18   85.29%
loadit>    Histogram-KG:                                 117   6   85.93%
loadit>    Histogram-KG:.........................        118  17   87.74%
loadit>    Histogram-KG:                                 119  16   89.45%
loadit>    Histogram-KG:                                 120  23   91.90%
loadit>    Histogram-KG:                                 121  25   94.56%
loadit>    Histogram-KG:.........................        122   7   95.31%
loadit>    Histogram-KG:                                 123  10   96.38%
loadit>    Histogram-KG:                                 124  15   97.97%
loadit>    Histogram-KG:                                 125   3   98.29%
loadit>    Histogram-KG:.........................        126   3   98.61%
loadit>    Histogram-KG:                                 128   3   98.93%
loadit>    Histogram-KG:                                 129   3   99.25%
loadit>    Histogram-KG:                                 132   2   99.47%
loadit>    Histogram-KG:.........................        134   2   99.68%
loadit>    Histogram-KG:                                 144   2   99.89%
loadit>    Histogram-KG:                                 145   1  100.00%
```

Field	Description
`KeyLen`	This is the length of the Identity Data after compression
`Count`	This is the number of records with that length
`Percent`	This is the cumulative percentage of the number of records having lengths less than or equal to the current `KeyLen`

Segment Lengths

The histogram can be converted into more useful data by running

```
%SSABIN%\histkg ReportFile
```

where `ReportFile` is the name of the log file containing the Table Loader output. It will produce a report similar to the one below:

```
Block Size      1792
Block Overhead     0
Index Name    idx_addr_fullkeydata
KeyData          225
CompLen          200

      key len    count      %   bytes   comp-1    comp-2   segs
           78        2   0.21     156      430       440      2
           81        4   0.64     324      860       880      4
```

```
          83       1   0.75      83      215       220      1
          85       6   1.39     510     1290      1320      6
          86       1   1.49      86      215       220      1
          87       4   1.92     348      860       880      4
          88       3   2.24     264      645       660      3
          89      14   3.73    1246     3010      3080     14
          90      10   4.8      900     2150      2200     10
          91      16   6.5     1456     3440      3520     16
          92       9  7.46      828     1935      1980      9
          93      17  9.28     1581     3655      3740     17
          94      19  11.3     1786     4085      4180     19
          95      15  12.9     1425     3225      3300     15
          96      32  16.31    3072     6880      7040     32
          97      31  19.62    3007     6665      6820     31
          98      29  22.71    2842     6235      6380     29
          99      40  26.97    3960     8600      8800     40
         100      45  31.77    4500     9675      9900     45
         101      25  34.43    2525     5375      5500     25
         102      31  37.74    3162     6665      6820     31
         103      28  40.72    2884     6020      6160     28
         104      34  44.35    3536     7310      7480     34
         105      46  49.25    4830     9890     10120     46
         106      30  52.45    3180     6450      6600     30
         107      34  56.08    3638     7310      7480     34
         108      40  60.34    4320     8600      8800     40
         109      21  62.58    2289     4515      4620     21
         110      34  66.2     3740     7310      7480     34
         111      37  70.15    4107     7955      8140     37
         112      48  75.27    5376    10320     10560     48
         113      27  78.14    3051     5805      5940     27
         114      28  81.13    3192     6020      6160     28
         115      21  83.37    2415     4515      4620     21
         116      18  85.29    2088     3870      3960     18
         117       6  85.93     702     1290      1320      6
         118      17  87.74    2006     3655      3740     17
         119      16  89.45    1904     3440      3520     16
         120      23  91.9     2760     4945      5060     23
         121      25  94.56    3025     5375      5500     25
         122       7  95.31     854     1505      1540      7
         123      10  96.38    1230     2150      2200     10
         124      15  97.97    1860     3225      3300     15
         125       3  98.29     375      645       660      3
         126       3  98.61     378      645       660      3
         128       3  98.93     384      645       660      3
         129       3  99.25     387      645       660      3
         132       2  99.47     264      430       440      2
         134       2  99.68     268      430       440      2
         144       2  99.89     288      430       440      2
         145       1 100        145      215       220      1
   Total 938                  99537   201670    206360    938

Keydata Offset    11
Key Overhead       4
Block Size      1792
Block Overhead     0

    compLen      bytes    segs    segs/key   DB-bytes   DB-blocks
          1    2090277   99537     106.116    2488425        1389
          2    1099868   49994      53.299    1299844         726
          3     770201   33487      35.700     904149         505
          4     605232   25218      26.885     706104         395
          5     506825   20273      21.613     587917         329
          6     441220   16970      18.092     509100         285
          7     394659   14617      15.583     453127         253
          8     360136   12862      13.712     411584         230
          9     332717   11473      12.231     378609         212
         10     311280   10376      11.062     352784         197
         11     293942    9482      10.109     331870         186
         12     279552    8736       9.313     314496         176
         13     267465    8105       8.641     299885         168
```

14	256088	7532	8.030	286216	160
15	247660	7076	7.544	275964	154
16	239544	6654	7.094	266160	149
17	233137	6301	6.717	258341	145
18	226822	5969	6.364	250698	140
19	221871	5689	6.065	244627	137
20	216280	5407	5.764	237908	133
21	212052	5172	5.514	232740	130
22	209622	4991	5.321	229586	129
23	205497	4779	5.095	224613	126
24	203016	4614	4.919	221472	124
25	198360	4408	4.699	215992	121
26	196834	4279	4.562	213950	120
27	193687	4121	4.393	210171	118
28	191040	3980	4.243	206960	116
29	189728	3872	4.128	205216	115
30	189150	3783	4.033	204282	114
31	187884	3684	3.928	202620	114
32	187668	3609	3.848	202104	113
33	185712	3504	3.736	199728	112
34	183654	3401	3.626	197258	111
35	181115	3293	3.511	194287	109
36	178472	3187	3.398	191220	107
37	176358	3094	3.299	188734	106
38	173478	2991	3.189	185442	104
39	173696	2944	3.139	185472	104
40	173280	2888	3.079	184832	104
41	173362	2842	3.030	184730	104
42	174840	2820	3.006	186120	104
43	176841	2807	2.993	188069	105
44	179072	2798	2.983	190264	107
45	180180	2772	2.955	191268	107
46	181302	2747	2.929	192290	108
47	181637	2711	2.890	192481	108
48	181016	2662	2.838	191664	107
49	179469	2601	2.773	189873	106
50	176120	2516	2.682	186184	104
51	174660	2460	2.623	184500	103
52	172656	2398	2.557	182248	102
53	169506	2322	2.475	178794	100
54	166352	2248	2.397	175344	98
55	164475	2193	2.338	173247	97
56	160208	2108	2.247	168640	95
57	158081	2053	2.189	166293	93
58	157092	2014	2.147	165148	93
59	157289	1991	2.123	165253	93
60	156160	1952	2.081	163968	92
61	155520	1920	2.047	163200	92
62	155390	1895	2.020	162970	91
63	156787	1889	2.014	164343	92
64	158424	1886	2.011	165968	93
65	160055	1883	2.007	167587	94
66	161766	1881	2.005	169290	95
67	163473	1879	2.003	170989	96
68	165352	1879	2.003	172868	97
69	167231	1879	2.003	174747	98
70	169110	1879	2.003	176626	99
71	170989	1879	2.003	178505	100
72	172684	1877	2.001	180192	101
73	174468	1876	2.000	181972	102
74	176344	1876	2.000	183848	103
75	178220	1876	2.000	185724	104
76	180096	1876	2.000	187600	105
77	181972	1876	2.000	189476	106
78	183652	1874	1.998	191148	107
79	185526	1874	1.998	193022	108
80	187400	1874	1.998	194896	109
81	188870	1870	1.994	196350	110
82	190740	1870	1.994	198220	111
83	192507	1869	1.993	199983	112

84	194376	1869	1.993	201852	113
85	195615	1863	1.986	203067	114
86	197372	1862	1.985	204820	115
87	198806	1858	1.981	206238	116
88	200340	1855	1.978	207760	116
89	200669	1841	1.963	208033	117
90	201410	1831	1.952	208734	117
91	201465	1815	1.935	208725	117
92	202272	1806	1.925	209496	117
93	202157	1789	1.907	209313	117
94	201780	1770	1.887	208860	117
95	201825	1755	1.871	208845	117
96	199868	1723	1.837	206760	116
97	197964	1692	1.804	204732	115
98	196234	1663	1.773	202886	114
99	193137	1623	1.730	199629	112
100	189360	1578	1.682	195672	110
101	187913	1553	1.656	194125	109
102	185684	1522	1.623	191772	108
103	183762	1494	1.593	189738	106
104	181040	1460	1.557	186880	105
105	176750	1414	1.507	182406	102
106	174384	1384	1.475	179920	101
107	171450	1350	1.439	176850	99
108	167680	1310	1.397	172920	97
109	166281	1289	1.374	171437	96
110	163150	1255	1.338	168170	94
111	159558	1218	1.299	164430	92
112	154440	1170	1.247	159120	89
113	152019	1143	1.219	156591	88
114	149410	1115	1.189	153870	86
115	147690	1094	1.166	152066	85
116	146336	1076	1.147	150640	85
117	146590	1070	1.141	150870	85
118	145314	1053	1.123	149526	84
119	144143	1037	1.106	148291	83
120	141960	1014	1.081	146016	82
121	139449	989	1.054	143405	81
122	139444	982	1.047	143372	81
123	138996	972	1.036	142884	80
124	137808	957	1.020	141636	80
125	138330	954	1.017	142146	80
126	138846	951	1.014	142650	80
127	139797	951	1.014	143601	81
128	140304	948	1.011	144096	81
129	140805	945	1.007	144585	81
130	141750	945	1.007	145530	82
131	142695	945	1.007	146475	82
132	143336	943	1.005	147108	83
133	144279	943	1.005	148051	83
134	144914	941	1.003	148678	83
135	145855	941	1.003	149619	84
136	146796	941	1.003	150560	85
137	147737	941	1.003	151501	85
138	148678	941	1.003	152442	86
139	149619	941	1.003	153383	86
140	150560	941	1.003	154324	87
141	151501	941	1.003	155265	87
142	152442	941	1.003	156206	88
143	153383	941	1.003	157147	88
144	153996	939	1.001	157752	89
145	154770	938	1.000	158522	89

The first section of the report summarizes the histogram.

Field	Description
Block Size	Database Block Size. Information only.
Block Overhead	Database overhead when storing records within a block including control structures and padding. Information only.
Index Name	The name of the IDX
KeyData	The sum of the length of the IDT columns (the Identity Data, uncompressed)
CompLen	`Compress-Key-Data(n)` value
KeyLen	The length of the Identity Data after compression
Count	The number of records with this `KeyLen`
Bytes	`KeyLen * Count`
Comp-1	Total bytes to store `Count` records with `KeyLen` using Method 1 (1 segment only)
Comp-2	Total bytes to store `Count` records with `KeyLen` using Method 0 (multiple segments)
Segs	The number of segments required to store `Count` records of `KeyLen`

The second part of the report gives space estimates for various values of `Compress-Key-Data`.

Field	Description
KeyDataOffset	Length of the Fuzzy Key including any partition
KeyOverhead	The overhead associated with storing the segment in the database
Block Size	Database Block Size. Information only.
Block Overhead	Database overhead when storing records within a block including control structures and padding. Information only.
compLen	`n` from `Compress-Key-Data(n)`
Bytes	The number of bytes required to store segments of this size
Segs	The number of segments used
Segs/Key	The average number of segments per IDX record.
DB-Bytes	The number of bytes for segment of this size (scaled up by `KeyOverhead`)
DB-Blocks	The number of blocks for segments of this size (based on the `Blocksize` and `BlockOverhead`)

To optimize performance, select the largest `compLen` value that minimizes `DB-blocks` and set `Compress-Key-Data` to this value (126 in the example above).

Large File Support

Some computer operating systems limit the maximum file size to 2GB. We will refer to these as "small systems". Other operating systems have native support for files larger than 2GB. We will refer to these as "large systems".

To overcome the file size limitation the Data Clustering Engine can combine the free space on a number of file systems into one large logical file . A logical file larger than 2GB is composed of a number of small files known as "extents ". Each extent must be less than 2GB in size.

Although this feature is not necessary on "large systems", it can be used to distribute the data over multiple file systems thereby making use of fragmented space.

Large File Support is designed for binary files such as the database and index files. Restrictions apply to its use for text files, as described later.

Directory File

The management of extents can default to internal rules or can be user-supplied. A file known as the "directory file " can be defined to specify the number of extents as well as their names and sizes.

Each large file's directory file is named using OS-specific rules. Currently the file, if present, is the name of the large file with `.dir` appended. For example, the directory file for `match.db` would be called `match.db.dir`. Directory files contain multiple lines of text. Each line contains two blank separated fields used to define an extent. The size of the extent (in bytes) is followed by the name of the extent. The maximum extent size is limited to 2GB - 1. An asterisk (*) may be used as a shorthand notation for the maximum extent size.

For example, these definitions define two extents. The first is limited to 1MB and the second extent defaults to 2GB - 1.

```
1048576match.db.ext1
*match.db.ext2
```

If all extents are to be of equal size, you can define a template for the base name of the extents. For example

```
048575match.db.ext
*
```

will allocate extents of size 1048575 and name them using the rules documented in the *Default Extent Names* section below. Note that the second line containing the asterisk enables this type of processing. Also, this mode of processing requires the extent size (1048575 in this example) to be a power of 2 (1048576 in this example) minus 1. To allow all extents to have the maximum size of 2GB-1 use:

```
*match.db.ext
*
```

To allow `all` large files to have maximum size extents, create a file called `extents.dir` with the following text:

```
*%f
*
```

Note: It is possible to set the maximum extent size using the environment variable, *SSAEXTENTSIZE*. Eg. *SSAEXTENTSIZE=256k* will limit the size of all extent to 256kB (minus 1).

Default Extent Names

If a directory file does not exist when a large file is opened, default rules are used to name the extents. Each extent size defaults to 2GB - 1.

The first extent has the same name as the large file. Second and subsequent extents are created by appending two characters to the file name. The extensions are named aa, ab, ac,. . . az, ba,. . . zz. This means that (1+26*26) extents are possible giving a maximum logical file size of 1.3TB.

Using the example above, the extents would be named

```
match.db
match.db.extaa
match.db.extab
...
```

Small System Rules

Small systems support large files using the rules above. Extents are defined using a directory file. If a directory file does not exist default sizes and names are used.

Large System Rules

Large systems have native support for files larger than 2GB. Operating systems such as Windows-NT 4.0, HPUX and Digital/Compaq Unix are in this category.

Large files do not use extents on these Projects unless a directory file is defined. In the latter case, extents are still limited to 2GB - 1.

Restrictions

Large file support was designed for binary files. In general, `text` files are not supported by the extent mechanism.

Text files do not default to use extents when a directory file is not present. Even if a directory file is defined and extents are used, correct results can not be guaranteed on every platform. If you wish to use large text files, you should use an operating system that supports them natively.

`POST` will create a text file if either the Trim or `CR OUTPUT-OPTIONS` are used. If neither is specified, the output is binary (fixed length records) and can therefore use the extent mechanism.

CHAPTER 4

Operation

This chapter includes the following topics:

- Environment Variables, 86
- Limitations, 89
- DCE Console, 90
- How to Create and Run Jobs, 98
- How to Run Clustering, 101
- Stopping and Restarting Clustering, 106
- Utilities, 110
- Batch Process - Command Line, 116
- Report Viewer, 117
- Troubleshooting, 118
- How To, 122

Environment Variables

Environment variables are used to control certain aspects of the Data Clustering Engine's behaviour. Most of them are set in various scripts, which are modified when the DCE is installed and/or initially configured. They can also be added or changed from the Settings window within the DCE Console, except the *SSA_XXHOST* and *SSA_XXPORT* variables, which must be set before the DCE Server is launched.

SSA_DCETHREADS

Used to define the maximum number of threads used by the clustering process. Refer to the *Utilizing multiple CPUs* section for more details.

SSACACHEOPT

Defines the internal workings of the database cache(For internal use only).

```
SSACACHEOPT=option-string
```

where `option-string` is a series of characters, one per option:

Option1: file buffering

d	default buffering
o	no buffering
b	buffer size set to SSABSIZE

Option2: cache allocation strategy

n	preallocate whole cache at startup
l	lazy allocation (demand based)

SSACACHESIZE

Defines the size of the memory cache established by SSA-DB. The default size is operating system dependent and is 4MB in almost all cases.

```
SSACACHESIZE=n[b|k|m|g]
```

where

n	Represents the number of allocation units
b	Blocks (2048 bytes)
k	Kilo-bytes (kB)
m	Mega-bytes (MB)
g	Giga-bytes (GB)

SSAEXTENTSIZE

Defines the maximum extent size when creating a Large File.

```
SSAEXTENTSIZE=n[k|m|g]
```

where

n	Represents the number of allocation units
k	Kilo-bytes (kB)
m	Mega-bytes (MB)
g	Giga-bytes (GB)

Note: The actual size of each extent is actually one less that the size nominated by this parameter. Refer to the *Large File Support* section for more details.

SSAJAVA

The name of the Java executable. Usually set to "java".

SSALDR_KEYTH

The Loader utility `LOADIT` automatically creates n key generation threads, where n is the number of CPUs available. You may override this value by setting an environment variable *SSALDR_KEYTH=n* .

SSALDR_RBSIZE

The Loader utility `LOADIT` uses this environment variable to control the memory usage of its Reader thread. For example,

SSALDR_RBSIZE=5000 allocates enough memory for 5000 input records (which is the default value for this setting).

This variable is also used to calculate the size of the key generation output queues. They are calculated as

```
SSALDR_RBSIZE / number_of_key_threads * 8
```

SSAOPTS

This environment variable is used for setting tracing and logging level.

The default value is +L which causes all possible program error messages to be logged to a file called `SSAWORKDIR/dce.dbg`.

Some times Informatica Corporation Technical Support personnel may ask you to set this to other values to help Informatica to assist you in any problem situations. Refer to the *Troubleshooting* section for further details.

SSAPR

When using the SSA-NAME3 Version 2 Standard Populations rules, the *SSAPR* variable needs to be set to the pr directory. This variable is normally set during the DCE installation procedure. If using a SSA-NAME3 population directory from another location, this variable can be set from the DCE Console's `Settings Environment Settings` dialog.

SSASORTMEM

Used to define the maximum amount of memory to be used by SSA-SORT. It accepts sizes in bytes/k-B/MB/GB.

```
SSASORTMEM=n[k|m|g]
```

SSAWORKDIR

The DCE Server uses the value of the *SSAWORKDIR* environment variable as its default working directory. All the files that the Server creates are placed in to this directory unless the Console Client or the Project definition override this setting.

SSA_CSHOST

The TCP/IP `host:port` address of the DCE Console Server.

SSA_CSPORT

The TCP/IP `port` number on which the DCE Console Server is listening.

SSA_RBHOST

The TCP/IP `host:port` address of the DCE Rulebase Server.

SSA_RBPORT

The TCP/IP `port` number on which the DCE Rulebase Server is listening.

SSA_SEHOST

The TCP/IP `host:port` address of the DCE Search Server.

SSA_SEPORT

The TCP/IP `port` number on which the DCE Search Server is listening.

SSA_RBNAME

The default value for the Rulebase name in format

```
sdb:file:path/filename
```

When the DCE Console is launched, it connects to the Rulebase Server listening on the address specified in *SSA_RBHOST* and tries to open the Rulebase named in this parameter. The file name extension `".db"` is automatically added to the given `filename`.

If this Rulebase does not exist, the user is prompted whether or not it should be automatically created at that point.

Note: This environment variable should contain the name of the Rulebase as seen on the computer running the Rulebase Server.

Note: You cannot use the same `filename` for the Rulebase and a Project to be stored in the same *SSAWORKDIR* as this will lead to a file naming conflict.

Limitations

The Data Clustering Engine's design imposes some limits. The maximum values for some components are documented here:

Field	Description
Database file size	32 GB with the default space management granularity of 8 bytes. Raising the granularity while creating the database raises the maximum file size correspondingly. Default Large File Support extends the size to 1.3TB
Database index size	8 TB
Records / logical file	4 G (= 2^{32})
Field length	32767 bytes
Record length	64 kB - 1 (in compressed form)
Cluster number	4 G - 1 (= 232 - 1; zero is not a valid Cluster number)
Partition Data	255 bytes per component field
SDF Line Length	255 bytes

Object Names (for Sourced Data)

When sourcing data from Oracle, DB2/UDB or other ODBC compliant databases, the object names of tables, indexes and columns must conform to the conventions of the host DBMS with the following additional restrictions:

- Only Regular Identifiers are supported
- Regular Identifiers must not exceed 32 bytes in length

This means the table, index or column name cannot be defined as a Delimited or Quoted Identifier. A Delimited or Quoted Identifier is surrounded by double quotes and contains characters which cannot be used with Regular Identifiers. The column name: Father's Country of Birth in SQL is structured as "Father's Country of Birth" and is regarded as a delimited or quoted identifier due to the use of an apostrophe and spaces in the name. Other characters used with delimited or quoted identifiers include commas, slashes, dashes etc.

To overcome these limitations create a view of the table using regular identifiers, which do not exceed 32 characters, and source from this view.

DCE Console

The DCE Console provides the user with centralized control of the various components that make up the DCE Project.

The Console is a client/server application.

The Console server is a non-interactive program, which would normally run on the machine where the database resides. When it is run, the Console server will establish its environment and then wait for clients to connect. Once one or more clients are connected, the server launches and monitors the progress of the various DCE programs at the request of these clients.

The Console Client is a Java GUI program. It can be launched on any machine which is connected via TCP/IP to the Console Server's machine.

Running the Console Server

Before running the Console Server, ensure that the license server is running. Refer to the instructions in *INFORMATICA IR PRODUCT INSTALLER* manual to start the license server.

On a Win32 Computer

An icon for the Console Server is placed in the Informatica's Products Program Folder by the Installation process.

- Click on this icon to start the server.

On a Unix Computer

Before running the Console server on a Unix computer, ensure that the DCE environment variables are correctly set. These are provided in the `env/dces` script.

1. To start the Console Server use this command:

   ```
   $SSABIN/dceup
   ```

2. To stop it use this command:

   ```
   $SSABIN/dcedown
   ```

Running the Console Client

This section provides information about running the console client.

On a Win32 Computer

When you finish the installation process, you see an icon for the Console Client in the Informatica's Products Program Folder.

- Click this **icon** to start the client.

On a Unix Computer

Once the Console Server is running, the Console Client can be started using this command:

```
$SSABIN/dceconc
```

Using the Console Client in Setup Mode

When the Console Client is run, it interrogates the Server to determine the Server's mode of operation. If the Server is in "setup" mode, the Client initiates the setup test process. You are prompted to supply the required information such as the Rulebase name and selection of the tests to be run. It is recommended that you run all the setup tests.

1. After you fill in each of the required fields, click **OK**.

   Console will then go through the steps involved in completing the installation testing of DCE. These include:

   1. creating and initializing a Rulebase
   2. running the standard tests

   The Setup mode serves to confirm that the DCE installation is working correctly. Upon successful completion, the Server and Client change to "normal" mode of operation. The Client can then be used to carry out normal DCE operations. There is no need to restart either the Client or the Server.

   On the other hand, if an error should occur, the Server remains in **setup** mode and the install process can be repeated if required.
2. After the test procedure is completed, click **Finish** to switch to the normal mode of operation.

Using the Console Client in Mode

When the Console Client is started it connects to the Console Server determined by the -h parameter. It then determines from the Server the mode of operation, either `install` or `normal`.

Having determined that the mode of operation is `normal` the Client opens the main console window.

Settings

This section provides information on how to set variables.

- Click the **Settings** button presents a list of user-settable variables. These variables are described below.

Option	Description
Server Work Directory	The name of the directory on the server's machine where output files will be placed. This field is mandatory. Note that this value can be set using the `-w` command line option when launching the Console Client.
Client Work Directory	This defines the name of the Work Directory to be used by Client programs. If specified, it must specify a directory which is accessible to the machine on which the Client is running. At present, this parameter is used only by the Relate and DupFinder Clients. So, if you are not planning to run one of these, then there is no need to supply this parameter.
Service Group Directory	SSA-NAME3 v2 (or later) users may leave this value blank. If a value is given, the Console checks that this directory exists.
Rulebase Server	The name or the TCP/IP address of the host where the Rulebase Server to be used during this session is running.
Port	The port number on which the Rulebase Server is listening.
Connection Server	The name of the host where the Connection Server to be used during this session is running. **Note:** This information is currently not used.
Port	The port number on which the Connection Server is listening. **Note:** This information is currently not used.
Search Server	The name of the host where the Search Server to be used during this session is running.
Port	The port number on which the Search Server is listening.

Verbosity Options:

Option	Description
Statistics	If selected, the log files will include statistics mainly used for performance tuning and/or troubleshooting.
Usage Summary	Select this option to produce database usage statistics mainly used for performance tuning and/or troubleshooting.
Server Trace	If selected the Console Server produces verbose output. This is for troubleshooting purposes and should normally be disabled.
Live Progress	If selected the Console Client monitors and shows the progress of the jobs as they are running on the server.

Option	Description
Automatically Open Job Window	If selected the Console Client opens a new window for messages from each job.
Environment Settings	Click this button and it will open a separate **Environment Settings** dialog in which you can set additional environment variables that might be used to control various DCE tasks. These environment variables are documented for each task as required. For simplicity the Console Client shows only those environment variables that have been set from this dialog. In other words all the environment variables that were set at the time of the Server start-up also exist.

All the above controls are used by the Console Server to serve requests from the Client. Therefore, you should take care to see that the values are correct.

- After making any required changes, click **Ok**. At this point the Main Console Window is displayed.

The Main Console Window

You may now make a selection from the various buttons to perform the desired task. The buttons are arranged in two groups. The row of buttons along the top of the Console window are associated with the various objects with which a user might want to work, such as Project, Rulebase, etc.

- Click one of these buttons causes a second group of buttons to appear down the left-hand side of the Console window. These buttons are associated with various actions that can be carried out on the object selected from the first button group. For example, if you click **Rulebase** then the possible actions will be **Select**, **Edit** and **Create** and three buttons will appear in the second button group to allow you to select the desired action. In addition, there is a group of four buttons at the bottom of the left-hand panel. These buttons are independent of the top row of buttons and provide quick access to some basic functions, such as the Settings dialog described above.

In addition to the buttons there is a menu bar. By and large, the options available using the menu bar mirror those available via the buttons mentioned above.

To the right of the second button group is the messages panel. This is a read-only area where Console will display any progress messages, error messages, etc.

Along the bottom of the window is the status bar. This contains the current settings for **Work Directory**, **Rulebase** and **Project**.

The Options Explained

This section provides information on the Options.

Project Options

New

Allows you to create a new Project from a `project.sdf` file, from a flat (exported) file or by cloning the currently selected Project. The Console Server then loads the Project definitions into the Rulebase. This new Project will then be added to the list of available Projects.

The parameters are as follows:

Project Name

The name of the new Project. This name must match the name specified in the Project definition file.

Note: Using the same name for the Rulebase and Project is not allowed. (The default Rulebase name is "rule".)

Definition File

The name of the Project definition file which describes the new Project.

Database Directory

The directory into which the database for this Project will be created. The name of the database file will be the same as the Project Name with the file name extension `.db` automatically added to the name. If this field is left blank, then the default DCE working directory, controlled by the *SSAWORKDIR* environment variable, will be used.

Maximum DB Size

Select the maximum database size for this Project. The default is 32 GB. Note that the created database will not allocate the chosen amount of disk space but the size will grow as required.

Cloning a Project is possible if the Project exists in the current Rulebase. You must supply a new Project Name and the Database parameter as when creating a Project from a `.SDF` file.

Importing a Project from a flat file requires that the flat file have been created using the Export option (explained below). You must supply the Input File name, Project Name and Database parameters as when creating a Project from an `.SDF` file. It is also possible to retain the Project name that was used in the exported Project if you check the Match Project Name selection. If a different Project name is used, it is necessary to load the Project to create the clustering file with the new Project name before the new Project can be run. An exported Project may contain run time information from the original Project, if runtime information is not desired during cloning, the option Import As Template should be checked to import only the Project definition information.

Select

You can select a Project from the current Rulebase. The selected Project will be used in any subsequent operations.

The parameters are as follows:

Project Name

The name of the Project to be used.

Edit

Opens the Project Editor that allows editing all Project settings. Refer to the *Editing a Project* section for more details.

Load

Activates a Project that was imported from a flat file. See the Export option below.

Reset DB

Replace the current Project database with an empty one. The results of any Clusterings that you may have run so far will be discarded.

Delete

Delete the current Project from the Rulebase.

Status

Displays the status of the current Project and allows the user to change it. The below table describes the meaning of each status selection:

Status	Description	Read	Upd	Del	Run
Build	Under construction - unusable	yes	yes	yes	no
Test	Uncontrolled - any use	yes	yes	yes	yes
Production	Relaxed Production	yes	yes	no	yes
Locked	Frozen production	yes	no	no	yes
Prototype	Secured prototype - read-only use	yes	no	no	no

Export

Export an existing Project and/or SSA-NAME3 v1.8 Service Group to a flat file. This is useful for example if you wish to transfer the Project and/or SSA-NAME3 v1.8 Service Group to another Rulebase. You may also want to use this option to create "templates" for your future Projects.

The parameters are as follows:

Output file

The name of the flat file which will contain the exported Project or SSA-NAME3 v1.8 Service Group.

Service Group

The name of the SSA-NAME3 v1.8 Service Group to be exported.

Run Clustering

Opens a window from which you can choose a Clustering step defined for the currently selected Project to be run.

Rulebase Options

Select

Allows you to select an existing Rulebase.

Note: Rulebase > **Edit feature** should be used only if so instructed by Informatica Corporation technical support. Making changes to the Rulebase may render it unusable.

Edit

Invokes the Rulebase Editor to edit the current Rulebase, defined by Rulebase Name and Rulebase Server.

Create

Allows you to create and initialize a new Rulebase.

Note: When creating a new Rulebase, you cannot share the Rulebase and Project names as this will lead to a file naming conflict.

Tools Options

Run Clustering

Runs a selected clustering.

The parameter is, Clustering definition name.

Relate

Runs the Relate client. Refer to the *Batch Search Client - Relate* section for detailed description.

DupFinder

Runs the Duplicate Finder client. Refer to the *Batch Search Client - DupFinder* section for detailed description.

Report Viewer

Runs the Report Viewer client. Refer to the *Report Viewer* section for detailed description.

Run Program

Runs a user-specified program on the server.

The parameters is as follows:

Command Line

The program to run, followed by its parameters.

Common Common Functions

The following describes the functionality provided by the four buttons **Server Status**, **Settings**, **View Logs** and **Clear Messages**.

Server Status

This button activates the Server Status dialog, which reports the status of the DCE server programs, the Rulebase and the database associated with the current Project.

Settings

This option will display the dialog containing the current environment of the client. This is the same dialog as the one presented when the Client is first started. You may make any required changes to the environment variables.

View Logs

Use this button to activate the Log Viewer. The Log Viewer allows the various output files produced by DCE to be viewed.

The Log Views displays the files in a tree layout with the file size rounded to the nearest kilobyte. If the file is empty, the file is marked with an icon resembling letter X and no file size appears.

The Log Viewer also gives the user the ability to delete individual logs as well as all the logs associated with the run itself.

See further information about the Log Viewer and its usage below.

Clear Messages

Click this button to clear the main message window.

Launched Jobs

This is a list of all the jobs launched during the current session. You can access more information about a particular job in the list by click the **Open** button or double-clicking the job details in the list.

When reconnecting the client to the console server, the list will display all the currently running jobs for all console clients using the same Rulebase.

The progress messages for each job are not displayed automatically when a Client reconnects. The user must select a running job from the list and click **Open** (or double-click the item). This will open the usual progress window.

Options

Open

Opens a status window for the selected job.

Delete

Remove the selected job from the list. Note: only completed jobs may be removed from the list.

Refresh

Refreshes the list with the currently running jobs for the same Rulebase.

Log Viewer

Every time a procedure such as `Clustering`, `Relate` or a user-defined Job is started, a Run Number is assigned to that run and all relevant information is stored in the Rulebase. This information includes the completion status and details of any output files created during the run. The Run Number is used to uniquely identify the run.

The Log Viewer provides the user with the ability to access the run information for previously run jobs.

There are two classes of Jobs: Project Jobs and Global Jobs.

- Project Jobs are jobs which are run against a particular Project, such as Relate.
- Global Jobs are jobs which are not run against a particular Project. These jobs either involve more than 1 Project (eg. `Clone Project`) or are responsible for setting up a Project (eg. `Create Project`)

Choosing The Run To Be Viewed

Select the type of Job, either Project Jobs or Global Jobs, using the radio buttons. If Project Jobs is selected, select the required Project using the dropdown list of Projects. If Global Jobs was selected, then the Project need not be selected. Now choose the job name from the dropdown list of jobs. User-defined jobs are identified by their user-assigned names. Other procedures, such as `Relate`, are identified by the procedure name surrounded by asterisks, eg. *Import Project*.

When the job has been selected a list of runs for this job will be listed on the left hand side of the Log Viewer. The runs will be sorted in ascending order, so the most recent run will appear at the bottom of the list. The title for each run consists of the date and time when the job started and the Run Number which was assigned to this run. Select the run in which you are interested.

A list of the output files created by this run will appear below the list of runs. The most recently created output file will be automatically displayed in the right hand pane of the Log Viewer. To view other files in the list simply click on the required file in the tree display.

Note: The Log Viewer will truncate log files larger than 960k for display.

Other Functions Provided By The Log Viewer

The options are as follows:

Option	Description
Delete File	Use this option to physically delete the currently selected file.
Delete Run	Use this option to delete all output files and run information for the currently selected run.
Refresh	Use this option to reread the run information from the Console Server. This option is useful if a job is currently running and you want to check if anymore output has been created.
Close	Use this option to close the Log Viewer and return to the Main Console Window.

Parameters

Project Name

Select a Project from the list of available Projects in the current Rulebase, to view the list of job names belonging to the Project.

Job Name

Select a job name from the list of available jobs in the selected Project, to view the run information.

Project Logs

Check this option to see the Project dependant logs.

Global Logs

Check this option to see the Project independent logs.

Run-Information

The user is presented with a run information list of the selected Project job. The user can at this time make a selection to view the relative step information.

Step Logs

The user is presented with a list of steps belonging to the selected job. The user can now view the run logs, error logs, output files (if any) of each step by selecting the desired option.

How to Create and Run Jobs

A job is a collection of steps, where each step refers to a unit of work that needs to be performed. Jobs can be easily created, modified or deleted using the Job Editor that can be launched from the Console Client.

Job Options

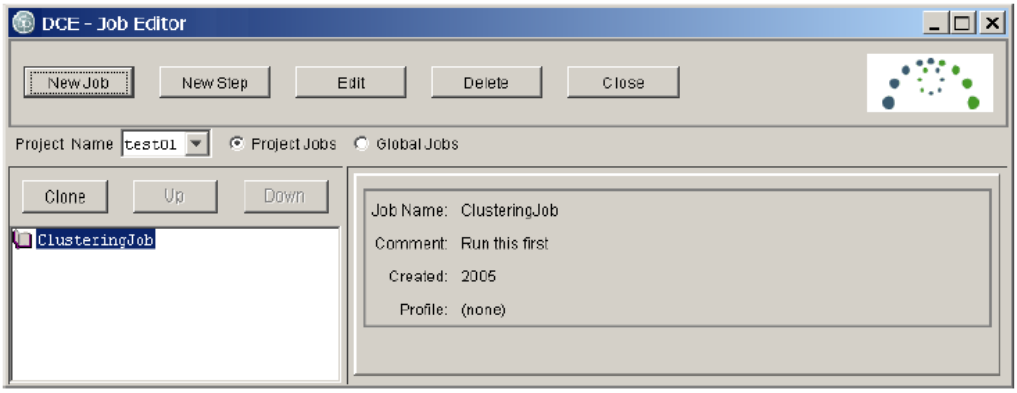
- Run

  This option allows the user to select and run a job, from a list of predefined jobs.
- Edit

This option allows the user to

1. Define a new job
2. Edit a predefined job; and also
3. Delete a pre-existing job.

Screen Shot:



New Job

- Allows the user to create a new job. This new Job will then be added to the list of available Jobs for the current Project. Jobs can also be global ie. not attached to any particular Project. To make a job global, select the appropriate radio button before you click the **New Job** button.

  Parameters

  Job Name

  The name of the new Job.

  Comment

  The user may specify a brief note about the purpose of the job.

New Step

- Allows the user to create and add new steps to the existing job. When the user chooses to add a new step a dialog will pop up with the list of available steps. A selection can then be made which will result in the respective input screen.

  Parameters

  Step Name

  The name of the new Step.

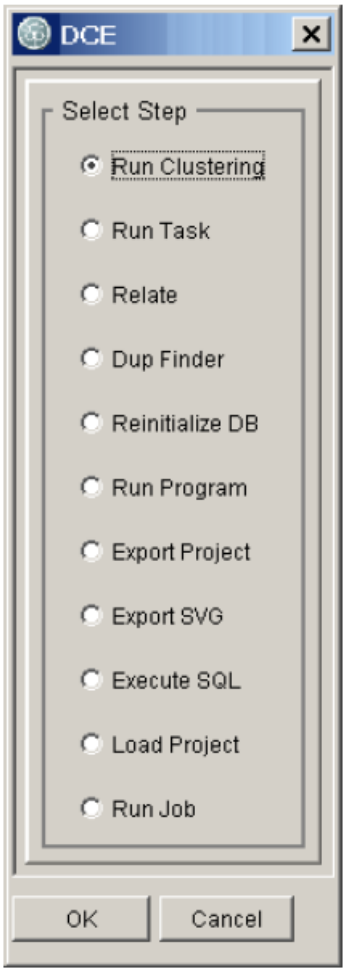
  Comment

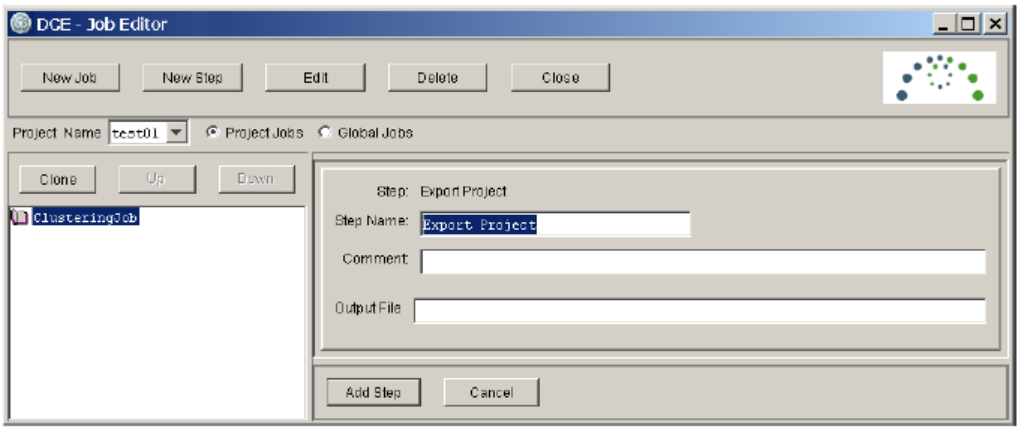
  The user may specify a brief note about the purpose of the step.

  Input Parms

  Input parameters required for the step.

Screen Shot:





- Edit - The user can either modify the Name, Comment or the Input Parameters of any step using this option.
- Delete - The user can either delete a job or step using this option.
- Clone - The user can clone an existing job or a step within a job.
- Up/Down - The order of steps within a job can be reorganised using the up and down buttons.
- Close - Use this option to close the Job Editor.

How to Run Clustering

The clustering process can be summarized by these steps:

- Requirements Analysis
- Launch the Console Server and DCE Console
- Create Project Definition (using a `.sdf` file or the Project Editor)
- Create the Project in the DCE Rulebase
- Select (and Load) the Project
- Run the Clustering Process
- Review Results
- Adjust parameters and rerun if required

Requirements Analysis

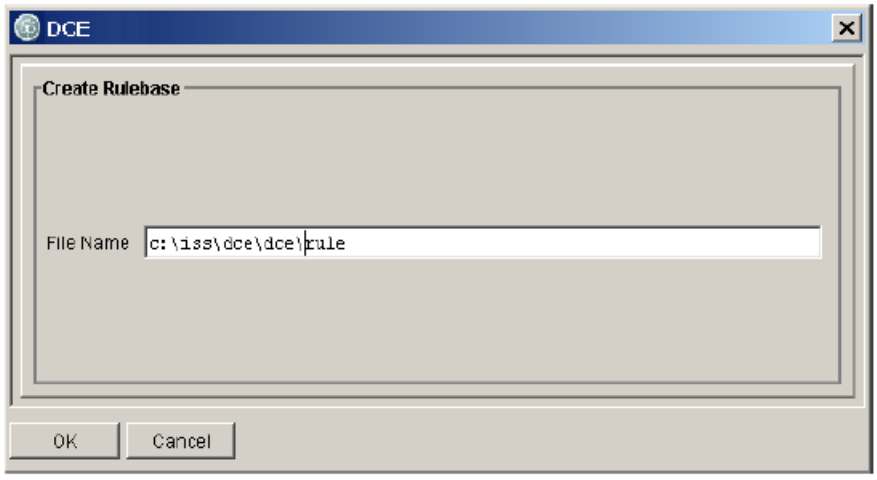
Determine business requirements. Define an objective and choose a suitable SSA-NAME3 population to implement that strategy.

Launch the Console Server and DCE Console

Launch the Console Server process on the machine where the server components have been installed. Refer to the *Running the Console Server* section for details.

On the client machine launch the DCE Console program. The first time you run the Console you will be prompted for a Rulebase name. The Rulebase will contain the rules for your Clustering Project(s) being used.

For example:



If you have an existing Rulebase you wish to use from a previous Clustering Project then specify it at this point. It is optional whether you specify the Rulebase filename with or without a file extension as an extension of `.db` will be automatically added to the Rulebase name.

Otherwise, enter the name of the new Rulebase and hit **OK**. You will be warned that the Rulebase does not exist but continue and you will be able to create the Rulebase within the main console window.

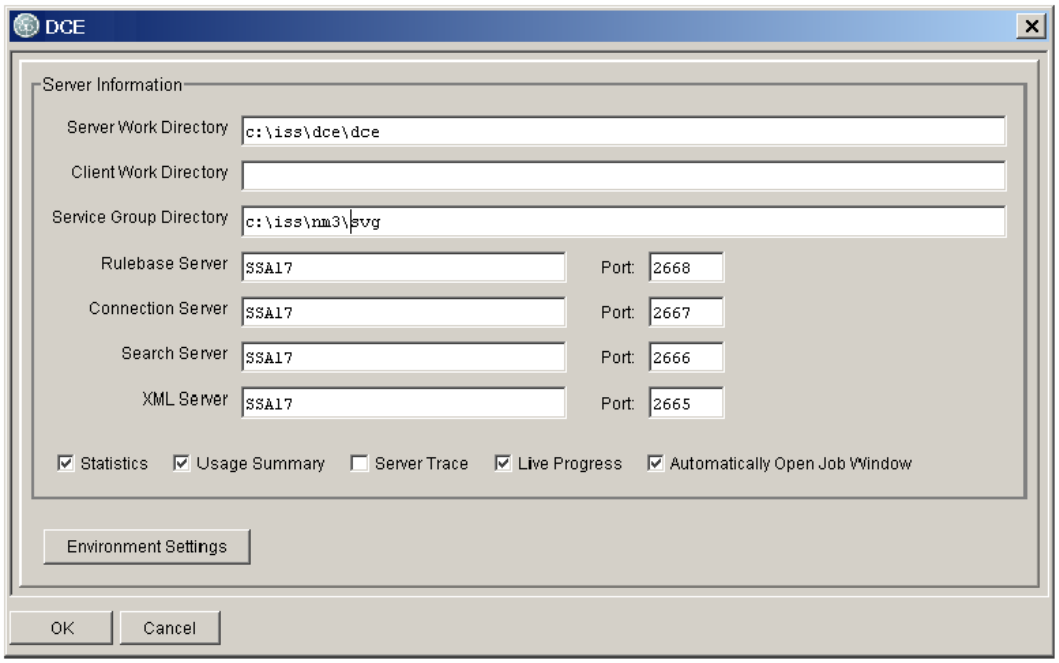
Note: When creating a new Rulebase, you cannot share the Rulebase and Project names as this will lead to a file naming conflict.

Create Project Definition

Create the Project Definition File using a text editor. It is useful to begin with a copy of an example Project and modify it to suit your own Project. You can also import an existing template Project and use the GUI Project Editor to adjust any settings.

Confirm Settings

At the Settings window confirm that the Server Work Directory and any other settings are correctly specified. It is important to note that the directory paths for the Server Work Directory and SSA-NAME3 v1.8 Service Group Directory (if used) are from the perspective of the Server machine. For example, if the Service Group is located on the E drive of the Server Machine, then it is referred to as E drive from the Console Client even if the local DCE installation was on the C drive.



The Server Work Directory is the location where the DCE will create the Rulebase, database and where the working files will be stored during the Clustering process.

The Environment Settings button will lead to a screen allowing further environment details to be specified. Refer to the *Environment Settings* section for further details.

Once you are satisfied that the correct details are entered then you can proceed into the main DCE Console window.

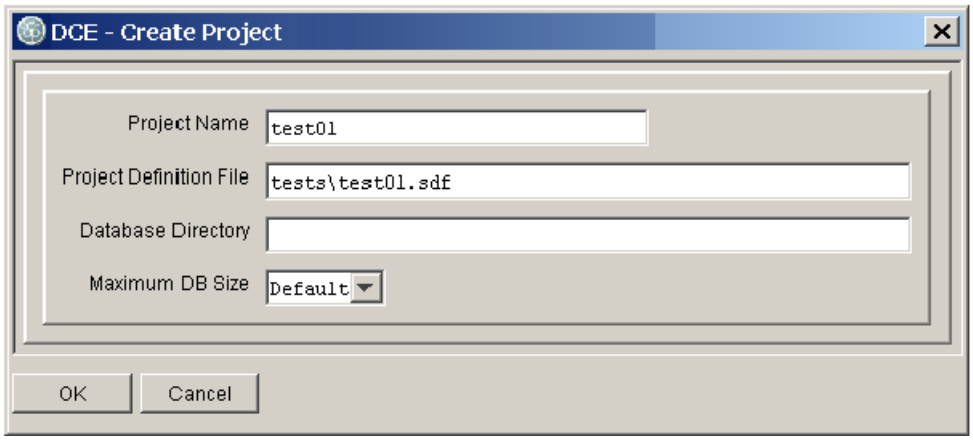
Create the Project

After you provide the required settings, you can create a project.

1. In the main DCE Console window, choose **New** under the Project heading to create your Project within the Rulebase.
2. Choose the appropriate option Create a Project from an SDF or Import a Project from a flat file.

Create the Project from an SDF

If you chose to use a Project Definition File (SDF file), you will be prompted for the name of this file, for example:

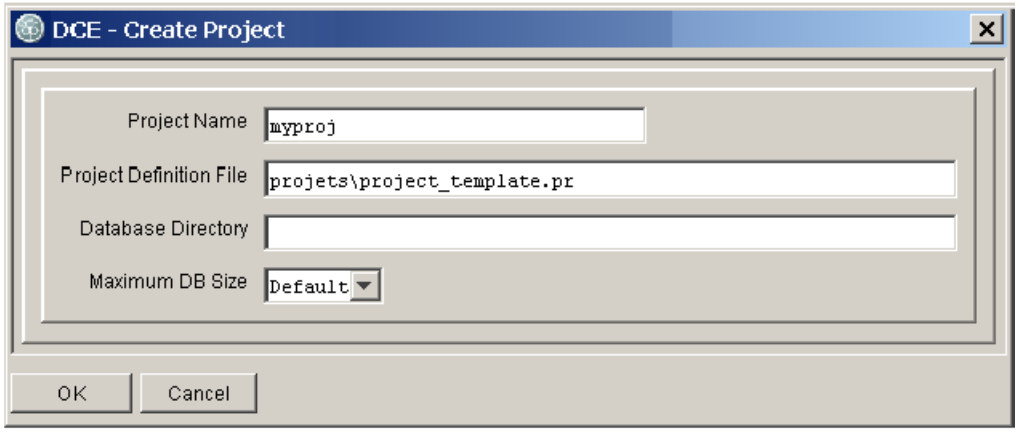


If the SDF file is located in a directory other than the Server Work Directory then the full path needs to be specified here. The Project definition file `test01.sdf` used in the above example is in a Server Work Directory subdirectory called `tests`.

- After you click **OK**, the Project is read and loaded into the Rulebase.

Create the Project by importing a template

If you chose to import a Project from a flat file, you will see this slightly different dialog:

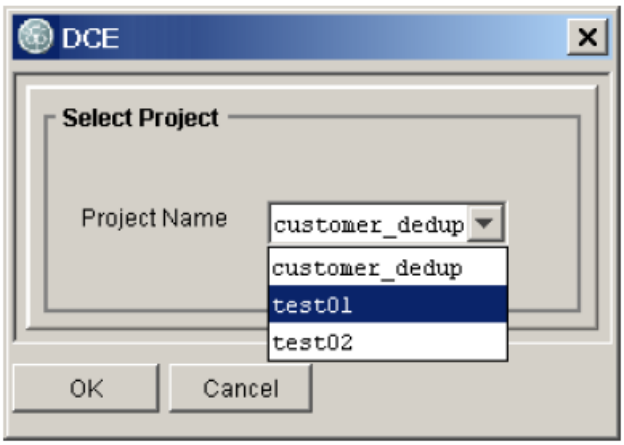


- After filling in the details, click **OK**, the Project (template) is read into the Rulebase. However the Project is not loaded at this stage, as it is assumed that you wish to change some settings using the Project Editor first.

Select (and Load) the Project

This section provides information on how to select and load the project.

1. Choose **Select** under the Project heading to select the Project you have just created. You will be able to select from all Projects which are a stored in the current Rulebase.
   For example, this particular Rulebase contains 3 different Projects:



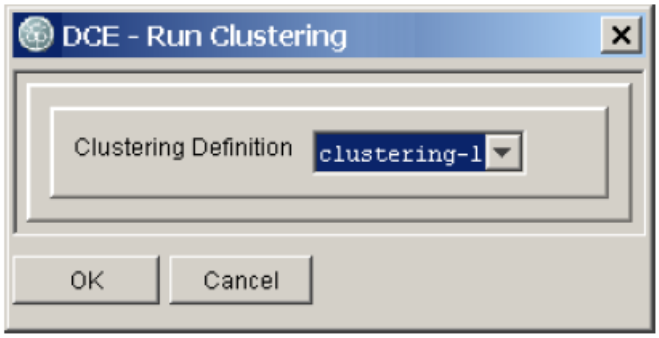
2. If you wish to change any Project settings at this stage, you may do so by starting the Project Editor from the **Edit** button. See further details in the *Editing a Project* section. After any changes and also if you have imported the Project from a flat file ("Project template"), you must load the Project. Click **Load** button to make the finalized Project `runnable`.

Run the Clustering Process

You now have a choice as to how you wish to initiate the Clustering Process. The choice will largely depend on how many Clusterings are contained in your Project.

Tools / Run Clustering

If you wish to run each Clustering separately or if your Project only contains one Clustering, then choose the Run Clustering option from under the Tools heading. You can then select which individual Clustering you want to run:

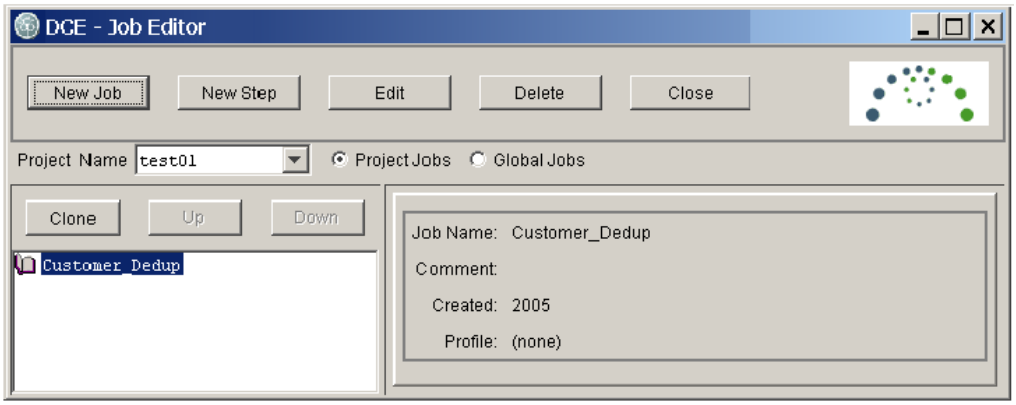


To run subsequent Clusterings, follow the same procedure for each.

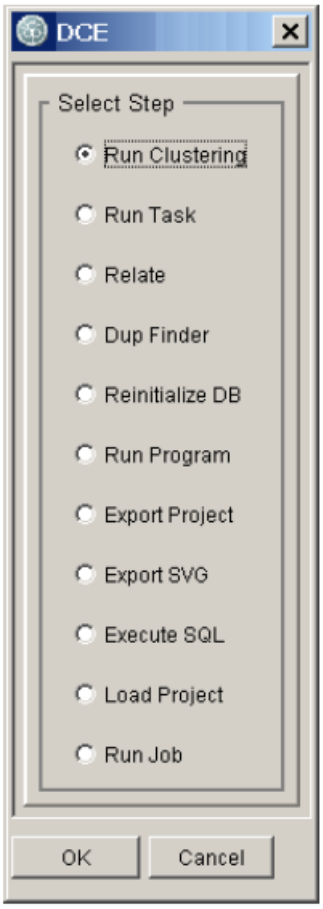
Create a Job

To run multiple Clusterings as a serial process, a Job can be setup to do so.

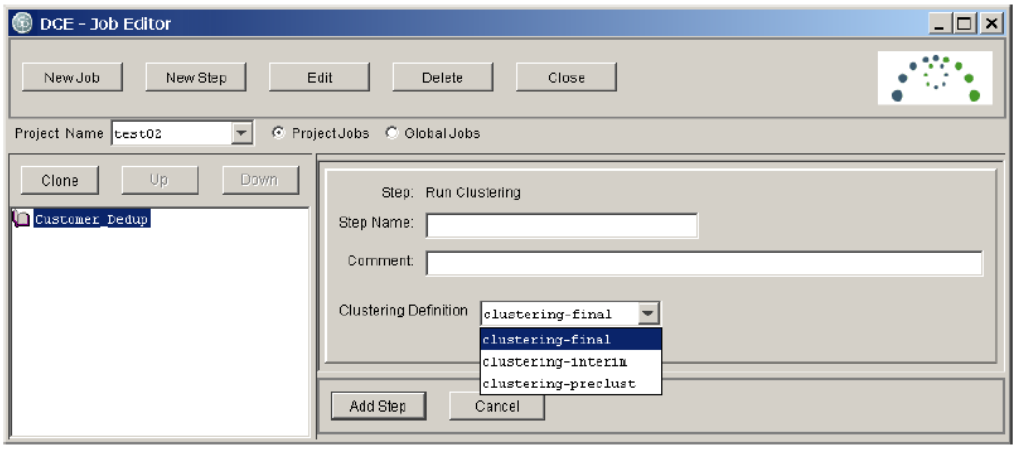
1. Select the **Edit** option from under the **Jobs** heading. This will invoke the Job Editor. Click the **New** Job button and type a name for the Job then click **Add** Job. You should see the new Job name listed:



2. Click the **New Step** button and Select the option to **Run Clustering** as follows:



3. Select the required Clustering.
   For example:



For each Clustering you wish to run, follow the procedure of adding steps to the job in the order that you require the Clusterings to be run.

4. Once all required jobs are added then close the Job Editor and select **Run** from the main Console window.
5. Choose the correct Job and click **OK**.

   For a complete explanation of various Job options see the *How to Create and Run Jobs / Job Options* sections.

Review Results

The Results of the Clustering Project can be viewed by selecting the Report Viewer from the Tools menu. Refer to the *Report Viewer* section for more details.

Adjust parameters and rerun if required

If you discover that some results are not satisfactory or wish to try a different scenario, you may use the Project Editor to adjust any Clustering parameters. After doing so you must reload the Project and reset the database to remove the old results before running the same Clustering steps again.

Stopping and Restarting Clustering

The clustering job may be stopped and/or restarted. The clustering process can only be stopped in between the steps or during the CLUSTER phase itself, not during other steps such as loading keys, sorting, etc. This can be useful in the following circumstances:

- While you are running a large clustering and you wish to take periodic backups of the database and index files
- You wish to stop the clustering process before it finishes so that you may view the intermediate results and assess its performance

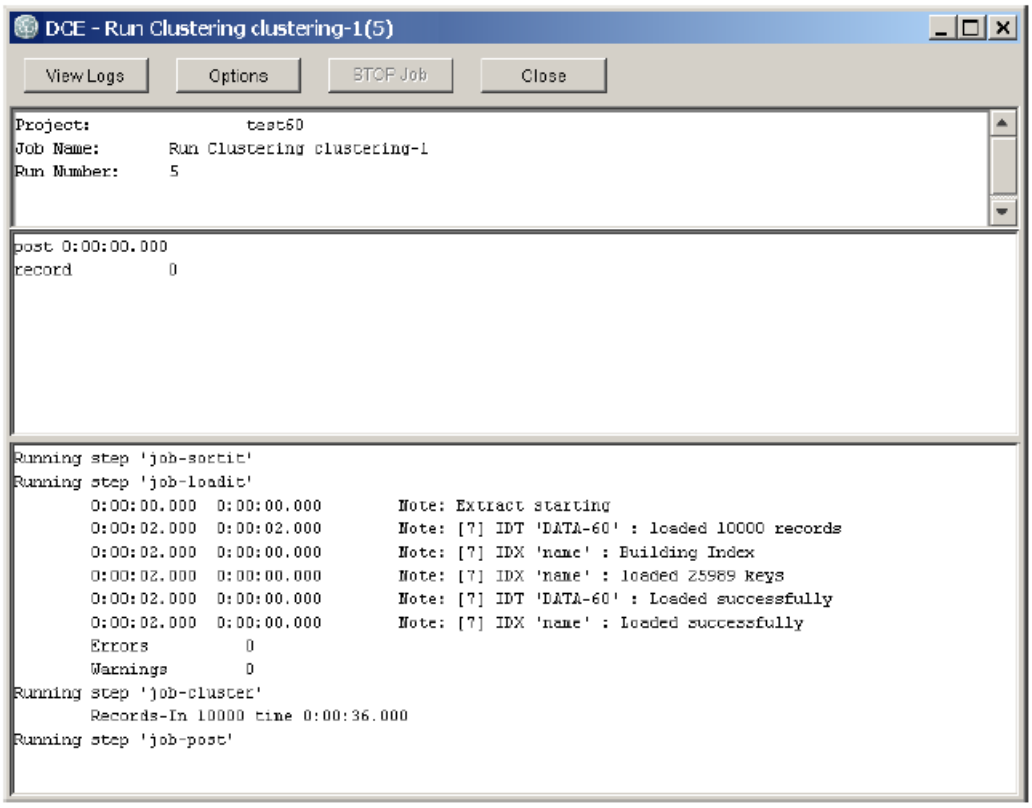
Stopping and Restarting Clustering Manually

The job can be interrupted manually and placed into a "" by using the relevant features available in the DCE Console window:

Monitor the Clustering Job

While the clustering process is running, make sure that the Clustering job progress window, such as below, is visible. If it is not visible, double-click the row that contains the details of the Clustering job in the Launched Jobs panel in the main console window.

Figure 1. Clustering Progress Window indicating "running" status

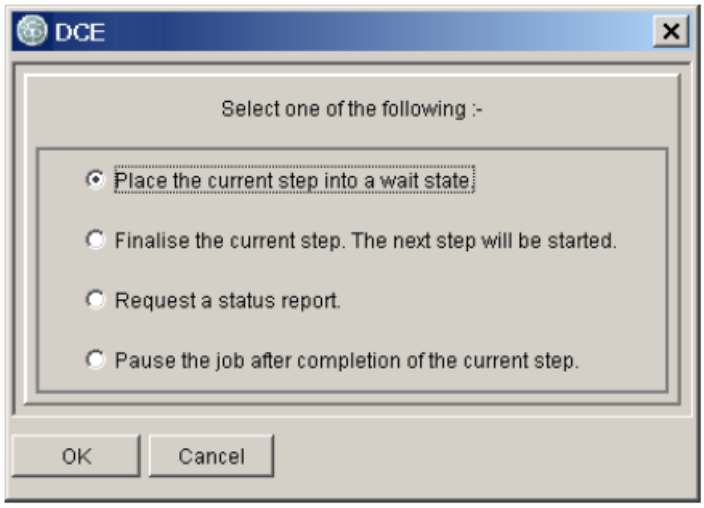


Pause the job

Click the **Options** button, choose "Place the current step into a wait state" option and click **OK**.

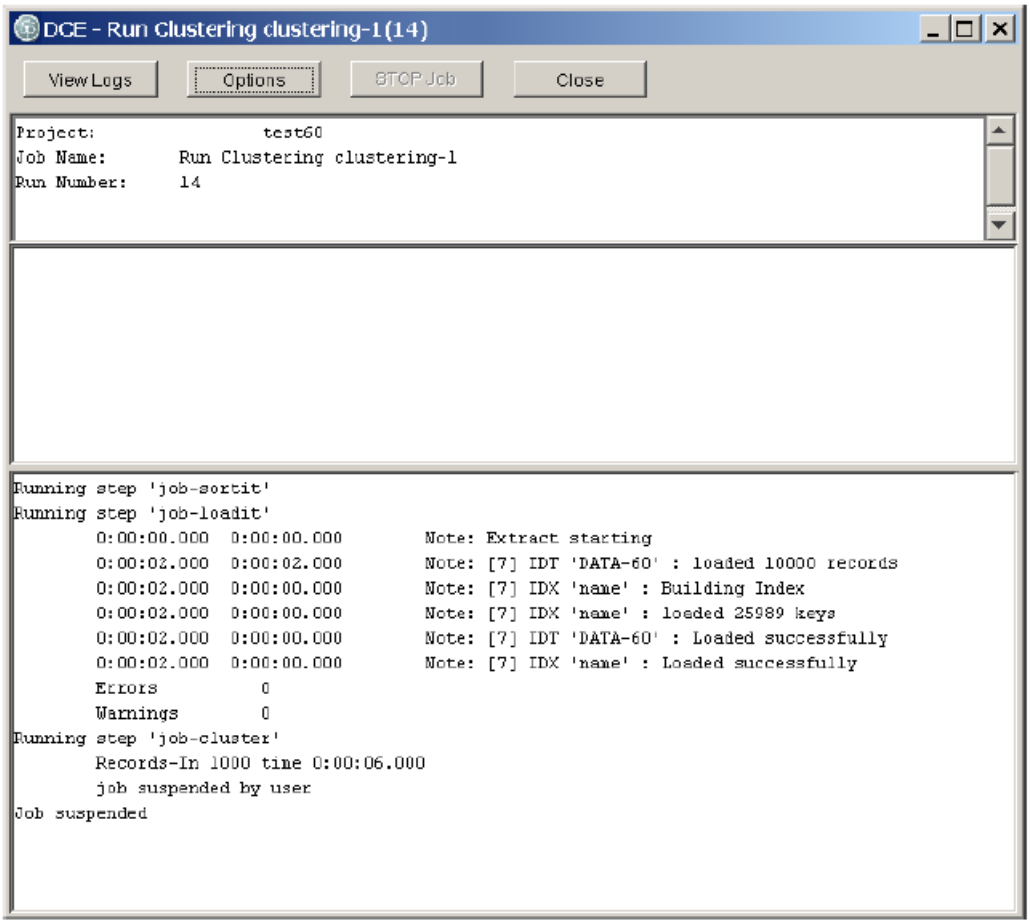
Note: If the Clustering process is not running when you click the **Options** button, only the option "Pause the job after completion of the current step is available".

Figure 2. Clustering Progress Window



Shortly the Clustering process enters the wait state, which is indicated by the "Job suspended" message in the progress window.

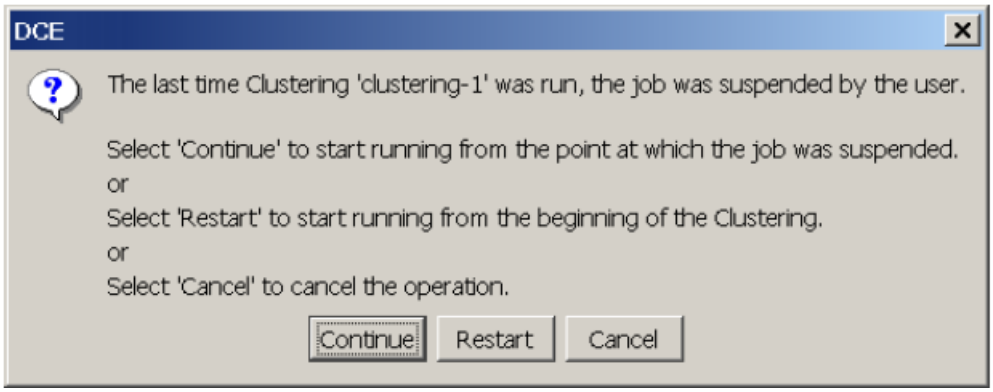
Figure 3. Clustering Progress window indicating "suspended" status



At this point it is safe to close the Console, shutdown the server and perform any maintenance tasks such as a backup. For backing up the Clustering work done so far by the suspended job refer to the Backing Up the Database and Index section.

To restart a suspended job

Restart the DCE Server and Console Client and select the **Project** that you were working on. Click the **Run Clustering** button and choose the suspended Clustering name from the list of available Clusterings. You are offered the following options:



- Click **Continue** to restart the Clustering from the suspended point.

  It is not possible to reliably suspend and restart a Clustering job that reads its input from a changing source, such as a SQL database. If you expect that you need to suspend such a job, then it is recommended that you extract the data temporarily to a sequential file and use this file as the input for the Clustering process.

Stopping and Restarting Clustering Automatically

The Clustering job will stop itself periodically when the Job parameter CHECKPOINT-TIME= has been specified in the SDF. When a checkpoint is to be taken, the job will save restart information and enter a 'wait' state, just as if the Wait option had been selected from the Console.

The job will stop when

- it is time for a checkpoint, or
- it has finished processing all of the input.

The sample Project in `test04.sdf` demonstrates this procedure.

Utilities

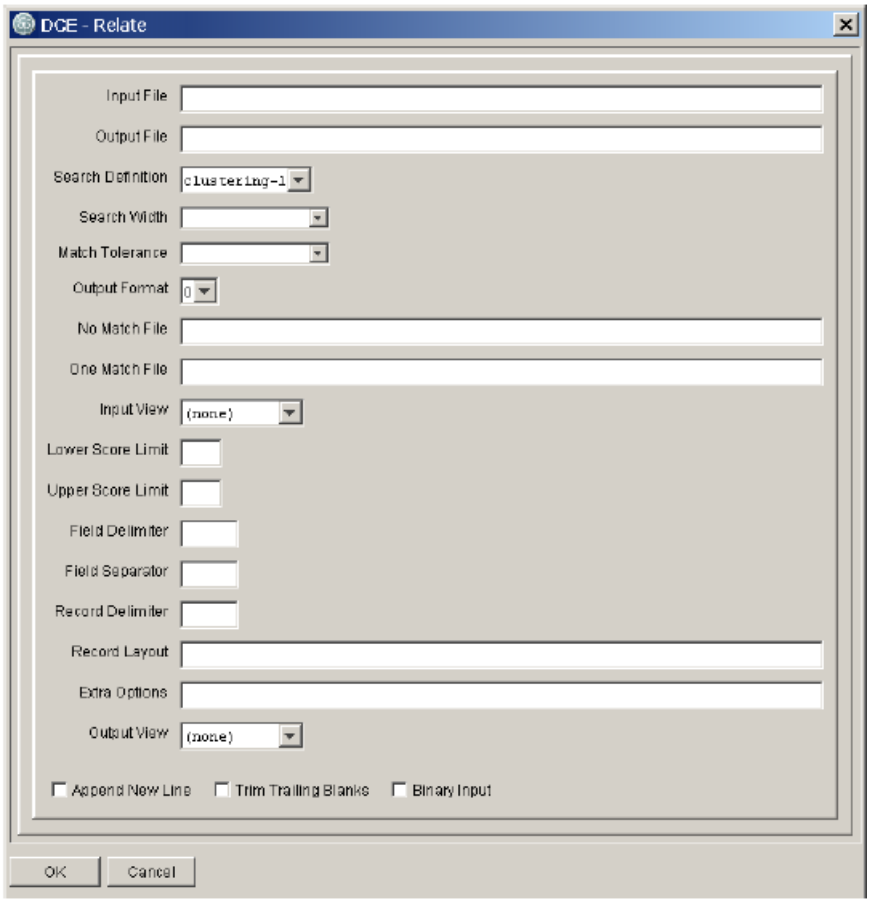
DCE Search Server
The DCE Search Server is a multi-threaded application that provides search and match facilities to the Relate and DupFinder Utilities.

Batch Search Client - Relate

The batch search client , Relate , is a batch search application which reads the search transactions from a flat input file, uses the nominated Search Definition to find matching records from the Clustering Database and writes the search results to a flat file.

Starting from the Console

Relate can be started from the Console Client by selecting **Tools** > **Relate**. This brings up the relate options screen:



Parameter	Description
Input File	The name of the input file. The records in this file must be separated by a newline unless the Binary Input option is checked. By default their format must match the layout of the clustering IDT to be searched. If the format differs from the clustering data layout, you must select an appropriate input view. This is a mandatory parameter. **Note:** The input and output files are read/written by the Client, so the files will generally be on the Client machine, or at least accessible from the Client machine.
Output File	Normally all records returned by the Search Server, i.e. those records that have an acceptable score as determined by the Search Definition, are written to the output file. If you also specify **No Match File** and/or **One Match File** (see below), then this file will hold all matching records not written to No Match and One Match files. This is a mandatory parameter.
Search Definition	You must choose the Search Definition to be used from this drop-down list.
Search Width	If you have predefined search widths (`Narrow`, `Typical` or `Exhaustive`) you can choose one here. Otherwise, if left blank, the control defined in the relevant search is used.
Match Tolerance	If you have predefined match tolerances (`Conservative`, `Typical` or `Loose`) you can choose one here. Otherwise, if left blank, the control defined in the relevant search is used.
Output Format	Choose the output report format from here. Values 0 - 7 are valid and are described below.

Parameter	Description
No Match File	Name of the output file to hold records that had no matches. This is an optional parameter.
One Match File	Name of the output file to hold records that had one match. This is an optional parameter.
Input View	If the input IDT layout does not match the clustering IDT layout, then choose the predefined layout from this drop-down list.
Lower/Upper Score Limit	You can alter the score limits defined in the Search Definition using the `Lower` and `Upper Score Limit` options. Default values are 0 for the `Lower Score Limit` and 100 for the `Upper Score Limit`.
Field Delimiter Character	Field delimiter character (Refer to the *Delimited Input* section for more details).
Field Separator Character	Field separator character (Refer to the *Delimited Input* section for more details).
Record Delimiter	Record delimiter character (Refer to the *Delimited Input* section for more details).
Record Layout	Record layout (Refer to the *Delimited Input* section for more details).
Extra Options	This field can be used to enter extra command line switches supported by future versions of the Relate program. Refer to the *Extra options for Relate and/or DupFinder* section for more information.
Append New Line	Append a newline to the output report after each record. This option has effect only on report formats 0, 1, 3, 4 and 6. Without specifying this option all the output records are written into a single line and the output should be treated as fixed length records.
Trim Trailing Blanks	Remove trailing blanks from each output record. This option has effect only on report formats 0, 3, 4 and 6. This option also implies `Append New Line` so that the boundaries between the output records are not lost.
Binary Input	Select this option if the input file is binary or if the input records are not terminated by newlines (ie. if the records are fixed length). If this option is chosen the record length must match the clustering IDT record length.

Report Formats

This section provides information on Report Formats.

0	Emit each returned record on a new line (if the Append New Line option was selected) . Eg:
	`JobN000005A ALGER COLLINS`
	`JobN000032A COLLINS`

1	For each record returned emit the search record, a "#", the search number, the score and the search result.
	`JobN000005A ALGER COLLINS#000033064 100 JobN000005A ALGER COLLINS`
	`JobN000005A ALGER COLLINS#000033064 099 JobN000032A COLLINS`

2	For each search, emit the search and its number, followed by the list of records returned by that search together with the search number and the score returned for that record.
	
	`JobN000005A ALGER COLLINS  #000033064`
	`  JobN000005A ALGER COLLINS  #000033064 100`
	`  JobN000032A COLLINS  #000033064 099`

3	For each search, emit the search number, then the search record and then a list of the scores and records returned.
	
	`********** 000033064 **********`
	`---JobN000005A ALGER COLLINS`
	`100JobN000005A ALGER COLLINS`
	`099JobN000032A COLLINS`

4	For each record emit the search number and the record, both on the same line.
	
	`00033064JobN000005A ALGER COLLINS`
	`00033064JobN000032A COLLINS`

5	For each search, emit the best record only. The format is the search followed by the record, both on the same line. `JobN000005A ALGER COLLINS JobN000005A ALGER COLLINS`

6	For each record emit 8 zeroes followed by the record.
	
	`00000000JobN000005A ALGER COLLINS`
	`00000000JobN000032A COLLINS`

7	Same as 5, but each record is preceded by the score returned: `100JobN000005A ALGER COLLINS JobN000005A ALGER COLLINS`

Delimited Input

Relate can read the input file if it contains delimiters. The field delimiter, field separator and record separator can be defined on the console. You can specify either a printable character or an escape sequence such as `\n` or `\x0a`.

The input data must be transformed into IDT layout (or into Input View layout if Input View is also specified).

The record layout is a comma-separated list of the following items:

- Field length (decimal)
- R/L justification (optional, L is the default)
- Filler character preceded by a dash (optional, default is a blank)

The following example defines two fields. The first is 30 bytes using the default justification and filler. The second field is 10 bytes, right justified and filled with 0.

```
30,10R-0
```

SQL Input

Relate can read input records from an SQL database instead of a file. In order to do this you must:

- Define source table(s) in the *UST* Section of the SDF using the `define_source` clause.
- Create a `Logical-File Definition` with `INPUT-FORMAT=SQL`
- Run Relate with the input file parameter set to "lfile=xxx" where `xxx` is the name of the Logical- File Definition.

The source definition should match the layout of the IDT (same field names, offsets and lengths). If it does not, specify an input view so that the Search Server will convert the input record into IDT format prior to searching. Note that a `define_source` clause automatically creates an input-view with the same name as the source.

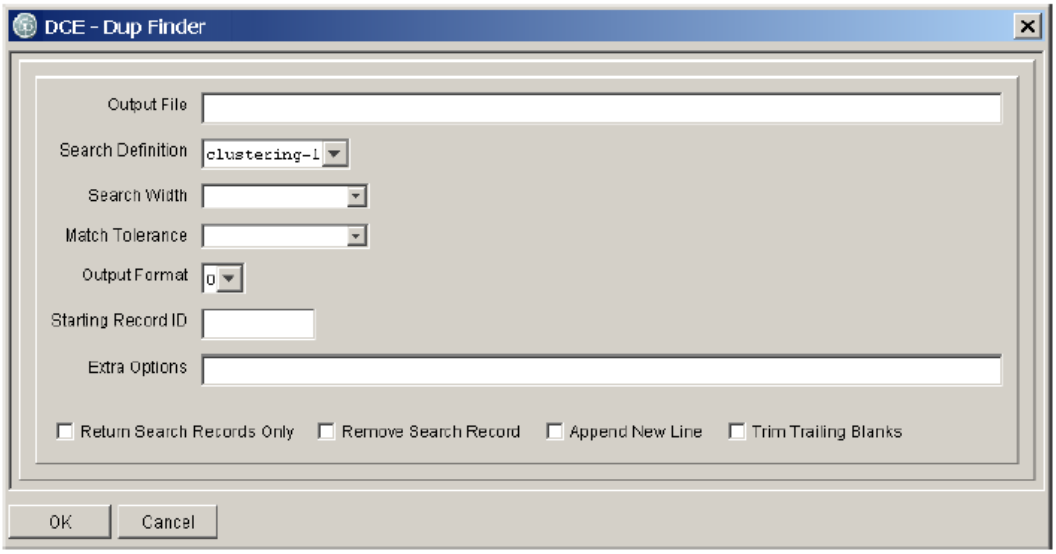
Batch Search Client - DupFinder

The DupFinder function is a batch search application designed to discover duplicate records within data previously loaded into the Clustering Database. It does so by using each record in the Clustering Database as a search transaction against the same database. It uses the nominated Search Definition to find duplicate records from the Clustering Database and writes the search results to a flat file.

Because every search transaction will have an identical record on the file, the report will display such matches unless the correct run-time option is used to remove the Source Record (see below).

Starting from the Console

DupFinder can be started from the Console Client by selecting **Tools** > **DupFinder**. This brings up the DupFinder options screen:



Field	Description
Output File	All duplicate records that have an acceptable score as determined by the Search Definition are written to the output file.
Search Definition	You must choose the Search Definition to be used from this drop-down list.

Field	Description
Search Width	If you have predefined search widths (`Narrow`, `Typical` or `Exhaustive`) you can choose one here. Otherwise, if left blank, the control defined in the relevant search is used.
Match Tolerance	If you have predefined match tolerances (`Conservative`, `Typical` or `Loose`) you can choose one here. Otherwise, if left blank, the control defined in the relevant search is used.
Output Format	Choose the output report format from here. Values 0 - 7 are valid and are described in the *Relate - Report Formats* section.
Starting Record ID	Enables commencement of the deduplication process at a nominated Record ID value.
Extra Options	This field can be used to enter extra command line switches supported by future versions of the DupFinder program. See the *Extra options for Relate and/or DupFinder* section below for more information.
Return Search Records Only	Only return the Search Record for which a match was found.
Remove Search Record	By implication of matching the same file against itself, the report will show matches caused by identical records. This is probably not desired so the search record can be hidden.
Append New Line	Append a newline to the output report after each record. This option has effect only on report formats 0, 1, 3, 4 and 6. Without specifying this option all the output records are written into a single line and the output should be treated as fixed length records.
Trim Trailing Blanks	Remove trailing blanks from each output record. This option has effect only on report formats 0, 3, 4 and 6. This option also implies `Append New Line` so that the boundaries between the output records are not lost.

Extra options for Relate and DupFinder

The following are the currently supported extra options that you can use in the Relate and DupFinder dialog box:

Option	Description
-n	Runs Relate and DupFinder in the multithreaded mode. For more information about the multithreaded Relate/DupFinder, see the *Threads* section.
-c<OutputView>	Nominates the name of the output view to format the records returned by the search. If you do not specify, records return in the IDT layout.
-y	Shows the layout of the chosen `OutputView` in the output reports. Applicable if you specify –`cOutputView`.
-b	Binary (fixed length) input file. The record length must match the IDT record length or input view length.

You can leave this field blank unless you have instructions from the Informatica Corporation technical staff.

Threads

Relate and DupFinder can run in a multi-threaded mode when the -n option is specified. Each search thread will independently connect to the Search Server and process searches in parallel.

There are two optional parameters associated with the -n switch: input queue and output queue. The syntax of the -n switch is as follows:

- `-nx[:y[:z]]`- use `x` search threads with an input queue of y records and an output queue of `z` records per thread.

The input queue specifies the length of queue that each thread will use to store the search records in. This queue must be long enough to allow the thread not to wait for I/O on the local relate input file. In general the default of 100 will be ample.

The output queue specifies the length of the queue that will hold each search thread's results. If any individual searches are expected to generate many matches, increasing the output queue size may improve performance.

Note: The output order of duplicate sets in a multi-threaded DupFinder report is dependent on the number of threads used to create the report.

Batch Process - Command Line

This section provides information on the command lines for Batch Processing.

bin\dcebatch

This script starts a Console job and waits for it to complete. The script indicates the completion status of the job by returning 0 for success or 1 for failure.

Make sure that `java` is in the current path. If not then it may be necessary to modify the `%SSATOP%\dce\dceenvc.bat`(Windows) or `$SSATOP/bin/dcebatch` (Unix) script to use the correct java executable.

```
set SSABIN=<DCE installation directory>\bin
%SSABIN%\dcebatch command [cmd args] options
```

Valid commands are:

```
Run job <Project Name> <Job Name>
```

<Project Name>

The name of the Project containing the job to be run.

<Job Name>

The name of the job to be run.

Valid options are:

-hcs<host>

Required. Console Server `host:port`

-w<workdir>

Required. Server Work Directory

-r<rbname>

Required. Rulebase Name (e.g `sdb:file:c:\InformaticaIR\dce\rule`)

-hse<host>

Optional. Search Server `host:port`

-hrb<host>

Optional. Rulebase Server `host:port`

-m-

Optional. Returns immediately after launching the job. The default is to wait for the job to complete.

-l{0|1|2|3}

Optional. Controls live progress messages.0=none, 1=messages, 2=progress, 3=both

-u<usgdir>

Optional. Server Name3 Service Group Directory

-v<[su]>

Optional. Server Verbosity setting, where s sets statistics and u sets usage summary. Refer to the *Verbosity Options* section for a description of this parameter.

Running from the Command Line

After defining a Job called `MyJob1` (Refer to the *How to Create and Run Jobs* section), start the job from the command line using this command:

```
%SSABIN%\dcebatch run job test01 MyJob1
   -hcslocalhost:2669
   -wc:\InformaticaIR\test1
   -rsdb:file:c:\InformaticaIR\test1\rule
```

where

localhost:2669

This is pointing to the Localserver.

c:\InformaticaIR\test1

This is the Work Directory.

sdb:file:c:\InformaticaIR\test1\rule

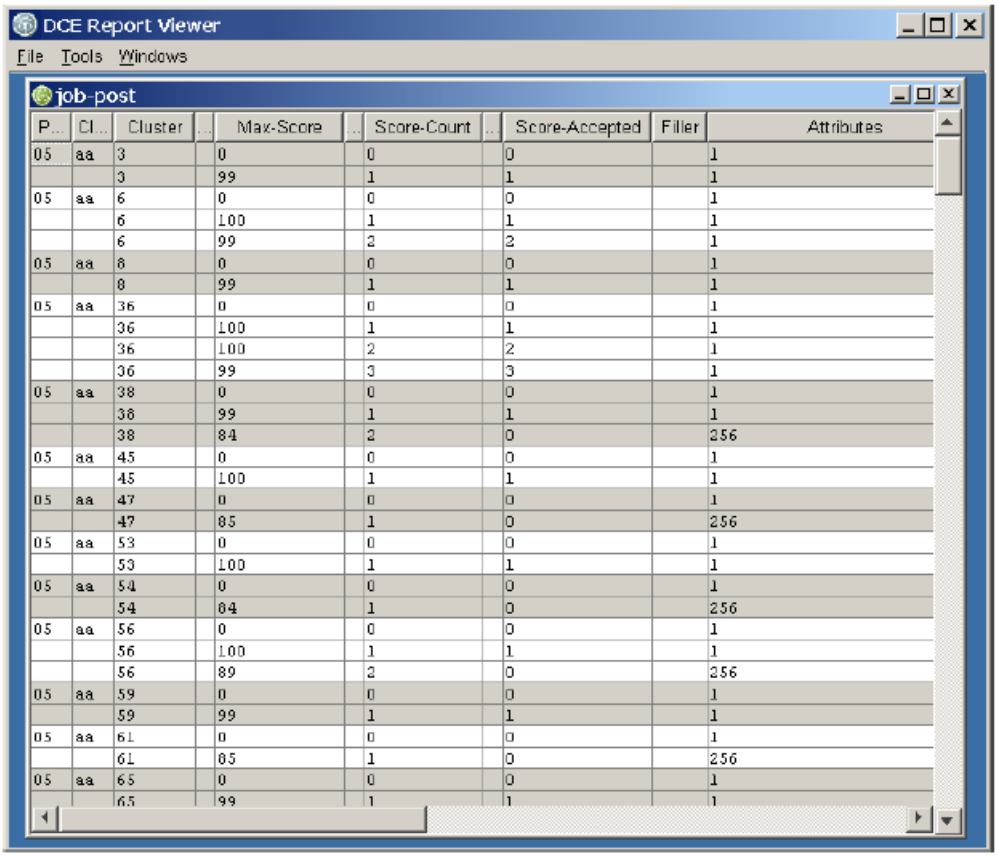
This is the Rulebase.

Report Viewer

The Report Viewer can be used to view the results of various Clustering runs.

Starting from the Console

Report Viewer can be started from the Console Client by selecting **Tools** > **Report Viewer**.



The **File** menu will allow you to open a **Post Report** or a **Database**. You can open the same one several times if you wish. This makes it easier to visually compare different parts of the same report at the same time.

Troubleshooting

The DCE includes a low-level tracing/monitoring utility `dumpshr`, which Informatica Corporation support staff might ask you to run in some circumstances. It is used to view the call stacks of all DCE server and utility programs. It is enabled by setting an environment variable of the shell that is used to launch the servers/utilities.

Since `dumpshr` uses shared memory to communicate with the servers and utility programs, it must be run on the same machine as the processes that it is monitoring. On Unix make sure that at least 15MB of shared memory can be allocated (`SHMMAX` kernel parameter).

Enable Tracing

1. Shutdown the DCE Server
2. On Win32 platforms:

```
set SSAOPTS=+t
```

On Unix platforms:

```
SSAOPTS=+t ; export SSAOPTS
```

3. Restart the DCE Server from the command prompt in which you set the *SSAOPTS* environment variable:

```
<DCE Installation Directory>\bin\ds
```

Disable Tracing

1. Shutdown the DCE Server.
2. On Win32 platforms:

```
set SSAOPTS=
```

On Unix platforms

```
unset SSAOPTS
```

3. Restart the DCE Server.

SSAOPTS environment variable

The following options may be set to enable trace and debugging features:

Option	Description
+t	Process/thread/stack tracing. Required for `dumpshr` operation. Also used to enable server stack trace for crashes (found in `dcexxsv.dbg`).
+L	Logs all error messages to `*.dbg` files. These files are either in the server work directory or in `/tmp`.
+u	logs process resource usage (threads, sockets, stack space, etc) to `*.dbg` files

Multiple options are enabled by concatenating them, eg. `SSAOPTS=+tL` (note these options are case sensitive).

Running dumpshr

Non-Interactive Mode

To run `dumpshr` in non-interactive mode, run

```
%SSABIN%\dumpshr -p
```

This will print the active program function stacks to the console (`stdout`). The output can be redirected to a file using normal shell commands, eg.

```
%SSABIN%\dumpshr -p >dumpshr.log
```

Interactive Mode

`dumpshr` can also be run in interactive mode to observe the interaction between server threads and utility programs. To execute in the interactive mode run command,

```
%SSABIN%\dumpshr >dumpshr.log
```

This will display a list of DCE processes. For each process, `dumpshr` displays the process id (`pid`) and the time it started (yyyymmddhhmmss). Each thread belonging to that process is summarized with a line showing:

- thread id (the main thread is 256)
- stack depth
- source module

- line number in source module
- function entry ("E"), exit ("X"), progress ("P")
- function name

The following commands are accepted:

Command	Description
Refresh Rate	Specifies how often the display is updated. The default is 10 seconds. To change the rate (in seconds), type one of the numbers '1' to '9' or '0' for 10 seconds.
Stack	A full stack for each thread can be displayed with the 's' key. This toggles between summary and full stack modes.
Print	The 'p' key will write a fully expanded stack to `stdout`. If you started the utility by redirecting the output the snapshot is written to the log file.
Quit	The 'q' key will exit the `dumpshr` utility.

Cleaning Up

On the Win32 platform, `dumpshr` uses Memory Mapped Files as its shared memory mechanism. Memory is allocated from the Windows swap file and is released when the last program using the memory terminates. As such, there is no special requirement to clean up.

On Unix platforms, `dumpshr` allocates shared memory as an IPC resource. DCE requires the kernel parameter `SHMMAX` to be set to at least 15MB. On Unix, shared memory is not released when programs using it terminate. This is a nice feature because any programs that core dump will leave their call stack in the shared memory. At a later stage, `dumpshr` can be used to display what the program was doing at the point of failure. However, the disadvantage of this is that when programs terminate abnormally the memory must be cleaned up manually. Shared memory can be deleted by running

```
$SSABIN/dumpshr -d
```

You can also delete IPC resources using the Unix utilities `ipcs` and `ipcrm`. DCE creates its shared memory and semaphore using the key `0x55A00001`. Future releases will increment this number when the layout of the shared memory changes.

Handling of Common Errors

This is a list of common errors encountered in DCE. This should help the user to resolve them without the need to call an Informatica Corporation technical specialist.

Rulebase is locked

The Console Client returns an error message such as this when trying to connect to a Rulebase:

```
An Error Has Occurred:
   Rulebase is locked
   Rulebase opened at Fri Feb 26 13:17:56 2010
   Server started at Fri Feb 26 13:17:46 2010
   Rulebase 'sdb:file:c:\a20i\dce\rule' is either corrupt or
in use by ssa.informatica.com IP=203.2.203.101 on port=2668
   IS ANOTHER RULEBASE SERVER RUNNING?
   ssacs_RulebaseStatus: rulebase_open 'sdb:file:c:\a20i\dce\rule' failed -20111109
```

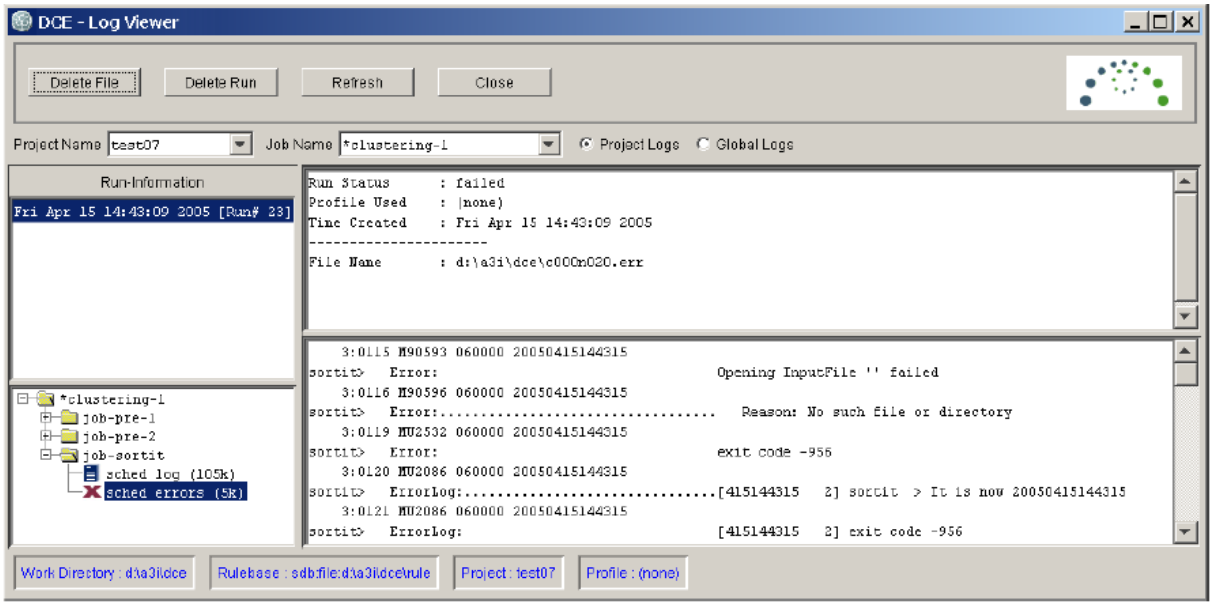
Rulebase is locked by another running Rulebase Server. If there's no other Rulebase Server running, then the Rulebase file is corrupted. This could happen if the DCE Server is shutdown incorrectly, either by the user or some other external occurrence like computer has been powered down abruptly.

Correction:

There are no repair utilities for corrupted Rulebases. The user must delete the present Rulebase files `rule.db` and `rule.ndx` and all other related Project files and re-start the Project from the beginning.

Opening InputFile ' ' failed

The Sort step fails with the message "Opening InputFile ' ' failed" followed by the message "Reason: No such file or directory".



The sort step cannot find the input file to sort. Usually this condition occurs when an option for the preceding step is missing.

For example, the Clustering Schedule calls for a job of type Pre before the Sort job but there is no option Format in the Clustering options list.

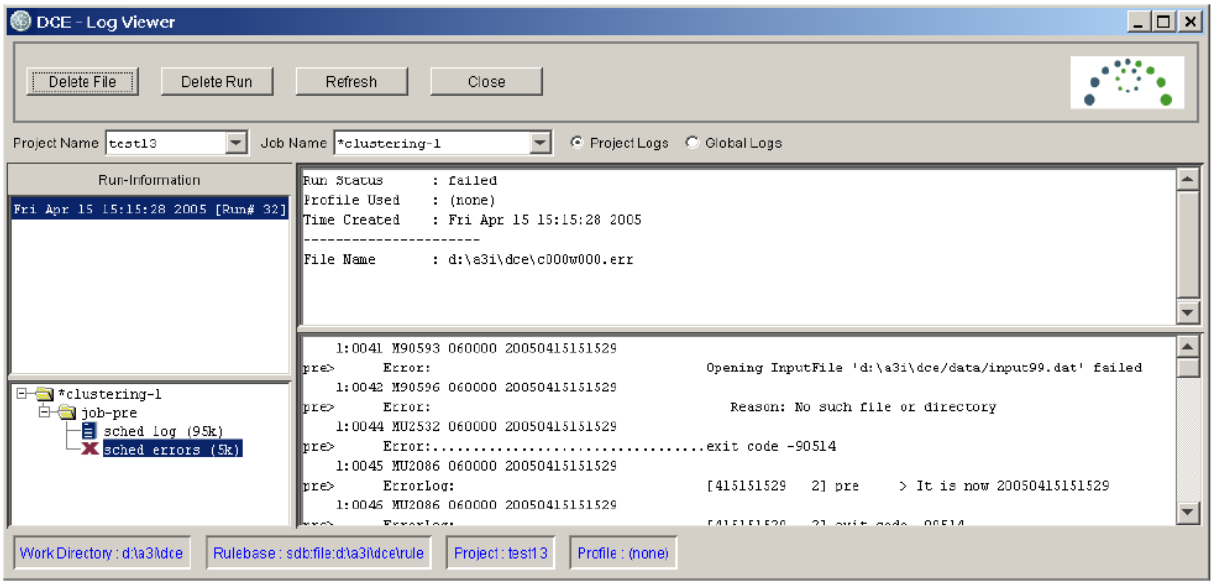
Correction:

- Check the Clustering options list for missing options.

Opening InputFile 'drive:<path><filename>' failed

The first step in a Clustering process fails to open the input file named in the message, followed by the message

"Reason: No such file or directory":



The input file as named in the message could not be found. The file name or path could be misspelled or the file could be missing.

Correction:

Check the path and name of the file and if the file exists.

How To

This section is intended to provide a "quick start" in terms of the parameters required to solve some types of business problems using the Data Clustering Engine.

It also contains hints and "tricks of the trade".

How to Re-Cluster Data

The Data Clustering Engine may be used to recluster records that have already been clustered. This can be achieved with either of the following techniques:

Technique #1

- Records are loaded into the database as preclustered.
- The records are reclustered using the same input file by specifying the `Merge` Clustering-method and the NO-ADD Job Option.

The NO-ADD option prevents the input records from being re-added and re-indexed on the database. It also prevents new cluster records from being added.

The real work is performed by the Merge option. The (re)clustering process will use records from the input file to match and score against records on the database. Records, which reach the scoring threshold, will have their clusters merged. The result is a reclustered file.

A set of sample definitions, which demonstrates this technique, can be found in `test02.sdf`.

Technique #2

- Records are loaded into the database and clustered using a key field such as Name.
- A key index is generated for a `new` key field, say Address.
- A second clustering step is run to recluster the records using the Address data. This will merge clusters of Names which have matching Addresses.

A set of sample definitions, which demonstrates this technique, can be found in `test03.sdf`.

Notice the following features:

- The `ClusteringID` must be the same for both Clusterings.
- Each Clustering's `Key Index` is named to avoid confusion. If `Key Index` was omitted, the same (default) file name would be used for each clustering.
- The second `LOADIT` job specifies `ReIndex` to rebuild the key index using the new `Key-Field (Address).`
- The Reclustering step specifies that the data is `PRE-LOAD`ed and that a `Merge` operation with `NO-ADD` is to be performed.

Input from a Named Pipe

On Unix platforms the input processor can read input from a named pipe. This means that it is possible to read data from another database without the need to create large intermediate files.

The concept is identical on all Unix platforms, although the command used to create a named pipe may vary between implementations. The following example is applicable to Linux.

```
mkfifo $SSAPROJ/inpipe
```

To use the pipe, specify its name as the Physical-File parameter in the `Logical-File-Definition` of the input file;

```
Logical-File-Definition
*======================
NAME=                 lf-input
PHYSICAL-FILE=        "+/inpipe"
COMMENT=              "named pipe for the load step."
VIEW=                 DATAIN
FORMAT=               TEXT
AUTO-ID-NAME=         Job1
```

For W32 platforms:

The pipe must be of the "blocking" type created by calling the Windows API function **CreateNamedPipe** before the Data Clustering Engine is instructed to read from the pipe.

To use a named pipe: You need to specify the name of the pipe in Microsoft's format, which is `\\server \pipe\<pipename>`. That is, two backslashes, the server name (or dot for the current machine), backslash, the word "pipe", another backslash, and then the name of the named pipe file.

Where the <pipename> part of the name can include any characters including numbers, spaces and special characters but not backslashes or colons. The entire pipe name string can be up to 256 characters long. Pipe names are not case sensitive.

If you don't specify something starting with `"\\.\pipe\"`, then an ordinary file will be assumed.

You can specify the file in the SDF. For example:

```
logical-file-definition
*======================
NAME=                   LF-input
COMMENT=                "named pipe"
```

```
PHYSICAL-FILE=          "\\.\pipe\namedpipe"
VIEW= DATAIN
FORMAT= TEXT
AUTO-ID-NAME=           Job1
```

Reformat Input Data

The `PRE` utility can be used as a standalone tool to reformat files. It can be used to

- reorder fields,
- delete fields,
- combine fields,
- insert text between fields.

The strength of this utility comes from its use of SSA-DB's view processor. An input view is used to describe the layout of the input file (`DATAIN11` in the example below).

The `Logical-File-Definition` describes the name and format of the input file; %SSAPROJ%/data/nm1k.dat and `Text` respectively.

`PRE` reads the input file using the input view and transforms the fields to match the output view specified by the Project-Definition's FILE= parameter. The output view is normally described in the file definition section of the SDF Under normal conditions, the output of `PRE` is a compressed binary file called `fmt.tmp`.

You can disable compression by specifying the Clustering-Definition's Options=--Compress-Temp parameter.

You can generate Text format output by specifying the Job-Definition's Output-Options=Trim,CR parameters.

```
Project-Definition
*=================
NAME=                   pre-job
ID=                     01
FILE=                   DATA11
DEFAULT-PATH=           "+"
*
Clustering-Definition
*====================
NAME=                   clustering-pre
CLUSTERING-ID=          aa
OPTIONS=                Format, --Compress-Temp
SCHEDULE=               job-pre
*
Job-Definition
*=============
NAME=                   job-pre
TYPE=                   pre
FILE=                   lf-input
OUTPUT-OPTIONS=         Trim, CR
*
*
Logical-File-Definition
*======================
NAME=                   lf-input
PHYSICAL-FILE=          "+/data/nm1k.dat"
COMMENT=                "the input file"
VIEW=                   DATAIN11
FORMAT=                 TEXT
*
```

The input and output views are used to specify how the file is to be modified;

- Fields are reordered by changing their relative positions in the input and output views.
- A field may be deleted by omitting it from the output view.

- Fields can be "combined" by reordering them to be consecutive. The input view for the next phase could then treat the adjacent fields as one "large" field.
- Fixed data can be inserted between fields by adding filler(s) to the output view.

Creating an Index and Search-Logic for any DATA field

The DCE utilities can be used to create an index and search any field in the IDT. Under normal circumstances the `KEY-FIELD` is used to generate name-keys using a Key-Logic module. This procedure can be modified to create an index for any field.

By defining an `IDX-Definition` and a `Search-Definition` which names the field to be indexed as the Key-Field and by specifying a Key-Logic and Search-Logic of User, we effectively define a key index and search definition that contains the exact key-value extracted from the DATA record.

For key building (`IDX-definition`):

```
KEY-LOGIC=User, Field(Phone)
```

For search (`Search-Definition or Clustering-Definition`):

```
SEARCH-LOGIC=User, Field(Phone)
```

Multi-clustering Data

You can use the multi-clustering functionality to combine results from several searches into one search. A multi-clustered data set may contain one or more clusters.

You can use the following techniques with multi-clustering to define a "household" clustering strategy that requires searches on name and addresses:

- Perform LOAD-IDT and build an index for Name.

  The following sample describes how to cluster by Name:

```
* ------------------------------------------------------------------------
* Clustering by NAME.
* ------------------------------------------------------------------------
clustering-definition
*====================
NAME=                           clustering-name
CLUSTERING-ID=                  AA
IDX=                            t3name
SEARCH-LOGIC=                   SSA,
                                System(default),
                                Population(usa),
Controls("FIELD=Person_Name SEARCH_LEVEL=Typical"),
                                Field(Name)
SCORE-LOGIC=                    SSA,
                                System(default),
                                Population(usa),
                                Controls ("Purpose=Person_Name MATCH_LEVEL=Typical"),
                                Matching-Fields("Name:Person_Name")
OPTIONS=                        Pre-Load
SCHEDULE=                       job-loadit
*
job-definition
*=============
NAME=                           job-loadit
TYPE=                           loadit
FILE=                           lf-input
*
```

- Perform second LOAD-IDT with ReIndex to rebuild the key index using the Address key field.

The following sample describes how to cluster by Address:

```
* ------------------------------------------------------------------
* Clustering by Address.
* ------------------------------------------------------------------
clustering-definition
*====================
NAME=                           clustering-address
CLUSTERING-ID=                  AA
IDX=                            t3addr
SEARCH-LOGIC=                   SSA,
                                System(testpops),
                                Population(usa),
                                Controls("FIELD=Address_Part1"),
                                Field(Addr)
SCORE-LOGIC=                    SSA,
                                System(testpops),
                                Population(usa),
                                Controls ("Purpose=Address"),
                                Matching-Fields("Addr:Address_Part1")
OPTIONS=                        Append, Pre-Load
SCHEDULE=                       job-ca-loadit
*
job-definition
*=============
NAME=                           job-ca-loadit
TYPE=                           loadit
OPTIONS=                        Re-Index
*
```

- Add a multi-clustering definition to create a household cluster with `Name` and `Address`.

  The following sample describes how to create a multi-clustering definition:

```
* ------------------------------------------------------
* MULTI-CLUSTERING by Name and Address
* ------------------------------------------------------
MULTI-CLUSTERING-DEFINITION
*======================
NAME=                           MULTI-CLUSTERING-nfs
CLUSTERING-ID=                  AA
IDT-NAME=                       DATA-100
CLUSTERING-LIST=                clustering-name,
                                clustering-address
SCHEDULE=                       job-cluster,
                                job-ca-post-plural-1,
                                job-ca-post-single-1,
                                job-post-all-1
*
```

Use the following rules when you work with multi-clustering definition:

- The Clustering ID must be the same for both clustering-definition and multi-clustering definition.
- Name the key index of each Clustering to avoid confusion. If you omit a key index, the default file name is used for each clustering.
- Ensure that the second LOAD-IDT job specifies ReIndex to rebuild the key index using the new key field, for example, Address.
- Invoke Clustering job step from multi-clustering definition and not from the individual clustering-definition. The following definition describes how to invoke clustering job step from multi-clustering definition:

```
MULTI-CLUSTERING-DEFINITION
*======================
NAME=                           MULTI-CLUSTERING-nfs
CLUSTERING-ID=                  AA
IDT-NAME=                       DATA-100
```

```
CLUSTERING-LIST=                clustering-name,
                                clustering-address
SCHEDULE=                       job-cluster,
                                job-ca-post-plural-1,
                                job-ca-post-single-1,
                                job-post-all-1
*

job-definition
*=============
NAME=                           job-cluster
TYPE=                           cluster
CLUSTERING-METHOD=              Merge
*
```

INDEX

.NET Framework 14

A

Accept limit 76

B

Batch DupFinder 29
Batch Processing 116
bin\dcebatch 116

C

checkpoint 110
CLUSTER 106
Cluster File 53
Clustering 9, 10, 73
Clustering job 110
Clustering Process 74
Clustering-Definition 27
Compressed Key Data 78
Connection String 57
Console Client
  Mode 90
  Setup Mode 90
Console Server 90
CONTROLS 76

D

Data Source Name 15
database
  testing 18
Database Directory 93
DCE 9, 15
DCE Console 90
DCE data type 55
DCE environment variables 90
DCE Identity Index 24
DCE IDT 46
DCE Loader 65
DEFINE_SOURCE 64
Denormalized Data 65
Denormalized-View 69
DNS 13

E

Edit Session 70
environment variable 77
environment variables 14
Environment variables 86

F

File Definition 46
Filler Character 47
Flattened Data 65
Flattening 65

G

graphical mode
  installation requirements 15

I

IDT 10
IDT layout 68
IDT parameters 23
IDT row 69
IDT-Definition 68
IDT-DEFINITION 23
IDX Definition 24
IDX Size 78
IDX-definition 122
Input File Processor 9
installation requirements
  X Window Server 15
installer 14

J

Java Development Kit 13
Java Runtime Environment 13
JAVA_HOME 14
JAVA_OPTS 14
Job Editor 98
Job Options 98
Job Steps 34
Join Expression 61
JRE 13

K

KEY-DATA 75
KEY-LOGIC 41
Key-Pre-Score-Logic 53
Key-Score-Logic 75
Key-Scoring 77

L

Large File Support 84
libssaoci9.so 14
Loader 69
LOADIT 11
Log Viewer 93
Logical-File-Definition 25

M

merge_clause 63
Monitor the Clustering Job 106
MULTI-SEARCH-DEFINITION 31

N

Native_DB_Service 15
Network Protocols 13

O

Object Names 89
ODBC driver 15
ODBC drivers 14
ODBC_DSN 15
Options 69
oracle ODBC drivers
  libssaoci9.so 14
  ssaoci9 14
Output Views 53

P

PARTITION 75
PATH 14
pre-clustering 11
pre-defined format 47
Project 10
Project Definition File 21
Project Editor 71
Project Options 93
Project-Definition 22

R

Re-Cluster Data 122
Report Viewer 117
Rulebase 9, 33, 93
Rulebase Options 93

S

Score Logic 44
Scoring 76
Search Definition 29
Search Server 53
Search-Definition 122
Search-Logic 41
Server Working Directory 73
Settings 91
SORTIT 11
Source Schema 54
source tables 64
ssaoci9 14
SSAOPTS 86
syntax 54
system requirements 13
System section 22

T

Testing Connectivity 15
Transform User-Exit 49
transform_clause 63
Transformations 49

U

Undecided Attribute 73
User Source Table 46, 54, 55, 57
User-Job-Definition 33
User-Step-Definition 34
USTs 61

V

Verbosity Options 91
View Definition 46, 49, 53
VirtualTable 57
Voting Attribute 73

W

Work Directory 93

X

X Window Server
  installation requirements 15