ERP Consulting

DataMagnet Report Description File (RDF)
Design and Construction Guide

August 12, 2008

Author's Note-

DataMagnet

You give it an unorganized report, tell it what you want, and, like magic, your data appears. A report that was once a disheveled display of customers, order numbers, and effective dates has suddenly become a detailed masterpiece - chock full of opportunities for analysis and number-crunching.

What makes the **DataMagnet** tick? How does it work? What makes it so versatile?

The answer lies in the RDFs – that is, the **Report Description Files**.

Report Description Files allow the **DataMagnet** to process various reports. They tell the program where to look for data in a report. They control how the data is picked up. They even decide where the data is written once the program has it in its grasp.

In this guide, we will take a look at nearly every aspect of RDF design and construction. We will discuss RDF anatomy as well as important design and construction ideas. After covering all these topics, we will describe the process of constructing an actual RDF.

Overview- Report Description Files (RDFs)

The purpose of a Report Description File is to define where useful data exists within a given report.

The RDF defines fields for each type of data in the report. Then, an RDF identifies all the possible line types in the report using a pattern match or data qualifier. When the program detects one of these lines, the RDF points to a set of coordinates that tell the program what data exists on that line, where the data is located, and what field the data should be written to. The program then picks up the data and writes it to the proper field.

An RDF has three main sections: the Data Line Marker section, the Data Coordinate section, and the Field section. It also contains four smaller sections: the Description Information section (DIN), the Report Identification section (RID), the Flag section (FLG), and the Report Line Format section. Each section is abbreviated within the RDF and has its own set of data fields. All RDFs are written in the following format:

```
RDF 0001S progname.p  @Company Name

DIN SECTION
[Type,Description]

RID SECTION
[FlagID,Start,Rule,Marker,Action,Length]

FLG SECTION
[ID,F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14,F15,F16]


RDF 01 progname.p

DLM SECTION
[Start,Rule,Marker,Group,Write,Action,Length,SeqGroup]

DLC SECTION
[Marker,Group,Seq,Field,Start,Length,Rule,Value,Action]

FLD SECTION
[Field,Output,Type,Order,DataLabel,Label,Action]

RLF SECTION
[Type,Description]

RDF 02 progname.p
```

(fig. 1)

Each section plays a different role in creating a "map" of the report. When written properly, an RDF allows the DataMagnet to find all of the desired data that the report contains. It also tells the program where undesired data is located and what elements it should ignore. An RDF defines where data is located, when and how the data is written, and how the data should be organized and labeled.

A separate RDF is needed to process each individual report. In some instances, however, more than one RDF may be needed to process the same report. Multiple RDFs can be created to accommodate reports that can be run in more than one way. In this case, the RDF contains multiple sets of descriptions where a set represents a given run option. For example, summary or detail.

Special rules and commands are often used in an RDF. These rules provide the programmer with more control over how the data is handled. These rules and commands can control how the data is picked up, when the data is written, and how the data will appear in the output file.

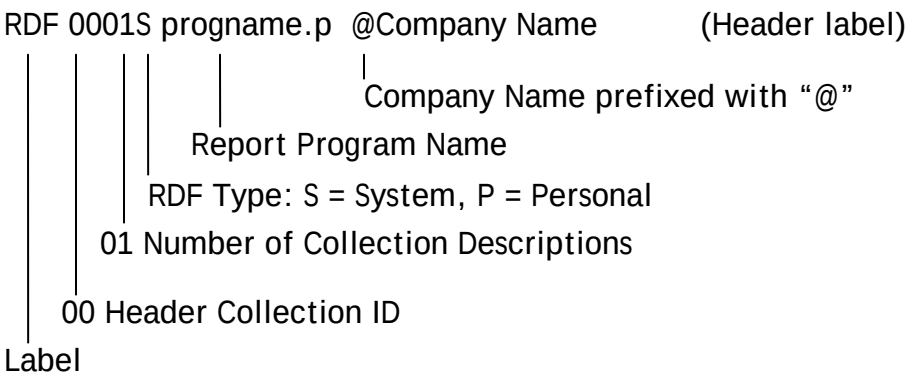
Because of its versatility, an RDF is capable of handling any report that contains repetitive properties and unique line types or qualifiers. Because nearly all reports fit this criterion, Report Description Files can be widely developed. RDFs are limited, however, by reports that lack repetition or have data with no definitive order.

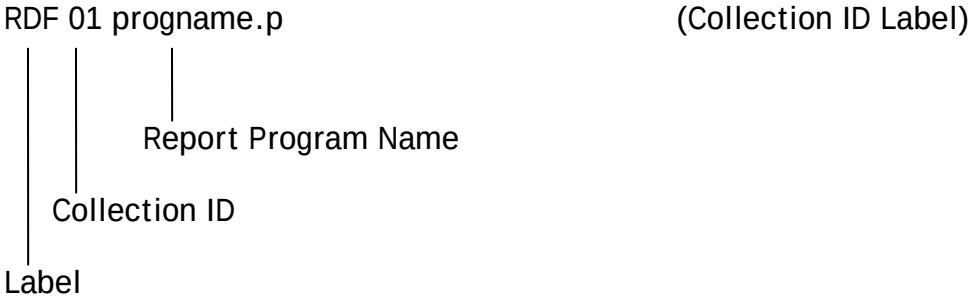
In the following chapters of this guide, the process of creating an RDF will be broken down. The first chapter, "RDF Anatomy", will describe the functions of each part of the RDF. This section contains an explanation and illustration of each of the RDF's 7 sections. The second chapter, "RDF Design", will discuss things to consider when designing a new RDF. This section talks about reading reports, defining line types, and incorporating special rules and commands into the RDF design.

RDF Anatomy

RDF Anatomy-

As discussed earlier, Report Description Files have seven basic sections. However, we are not going to cover any sections but rather discuss the RDF labels that also appear in the Report Description File.
With the exception of the first RDF label these labels identify the beginning of a collection of sections (DLM, DLC, FLD and RLF) that define an individual report layout. The first RDF label is the header label and contains some additional information used by DataMagnet.





Quick Tip:

The Label (RDF), Collection ID(00), and RDF Type(S) will not change and remain constant for all Headers. The Number of Collection descriptions will change only if it is a multiple RDF. The Report Program Name and Company Name will always need to be set. The RDF Type is set to "P" by the DataMagnet when saving a Personal RDF.

DIN Section-
The DIN section was created to help identify and track the RDF. For example, a report may differ slightly between versions of a particular system. Therefore, the RDF may contain subtle differences for each system version.
We track this by adding the system and version to the "SV" Type in this section. Below is an example of a completed DIN section.

```
DIN SECTION
[Type,Description]
DT,Created On:  10/06/04
SV,MFGPRO Version:  eB
DM,DataMagnet Version:  1.013b
LS,Licensed To: Starship Enterprises
```

(fig. 2)

The DIN Section contains only 2 fields, a Type and Description. Currently, there are only 6 information types, they are:

DT – Date the RDF was created.
SV – System and Version used by the client/company.
DM - DataMagnet version the RDF was designed on.
LS - Company the RDF was licensed to.

Quick Tip:

This information can be viewed by selecting About RDF from the DataMagnet's Help menu.

DLM Section-

Before DataMagnet can pick up any data, it has to be able to distinguish one line from another. That's where the Data Line Marker (DLM) section comes in. This section contains a pattern or data qualifier that represents every line type in the report. This allows the program to tell the difference between an Order line and an Invoice line, for example. Once it identifies the line, the program looks at the line type's **Action**. The **Action** directs the program what to do next Skip, Stop, or Pickup data. Actions other than skip or stop point the program to a set of coordinates in the DLC that tells it what data to pick up on that line. Here's an example of a completed DLM section:

```
DLM SECTION
[Start,Rule,Marker,Group,Write,Action,Length,SeqGroup]
1,=,glctrrp,0,0,skip
1,=,Page,0,0,skip
80,=,-----,0,0,skip
80,=,Period,0,0,skip
1,x,End of R,0,0,stop,0
27,=,Reporting,0,0,D001
27,=,Exchange,0,0,D002
1,=,CC,0,0,D003
51,p,bsssssssssssssssssssssssbbbbooooooooooooooon,1,1,D004
```

(fig. 3)

There are 8 data fields for each line in the DLM section: [Start, Rule, Marker, Group, Write, Action, Length and SeqGroup].

- **Start** refers to the column position that the line pattern or qualifier begins in.
- **Rule** determines if the line will be matched to a pattern or to a qualifier. If the Rule is set to **"="**, the line will be matched to the qualifier in the **Marker field**. If the Rule is set to **"p"**, the line will be matched to the pattern found in the **Marker field**. If the rule is set to "**x**", the line will be scanned to see if the marker exists within the range set by the length setting.
- **Marker** can be either a qualifier or a pattern. A qualifier is a hard-coated piece of data from the report. A pattern matches line types based on blanks, characters, and numbers located in certain positions.
- **Group** sets the group for data identification in the DLC section.
- **Write** sets the write level switch on the line.
- **Action** is a command to perform when marker is matched on a line.
- **Length** is the length of the Report line to match for the "**x**" rule.
- **SeqGroup** is the flag to set group sequencing on.

Qualifiers, patterns, grouping, and write switches are all covered in detail in the design section of this guide.

Action describes the action that will be taken on the line once it is identified. There are four possible actions: **skip**, **stop**, **D999- " line marker code "**, and **S999- " sequence marker code "**.

- **Skip** tells the program to ignore the line.
- **Stop** tells the program to stop processing the report.
- The **D-marker code** identifies line types.
- The **S-sequence marker code** identifies sequence lines types.

Now, let's put a line together-

Here are two lines from the screen shot:

27, =, Reporting, 0, 0, D001

1, x, End of R, 0, 0, stop, 0

The first example tells the program that the line will be identified as line type D001 if the qualifier, "Reporting", is found starting in column position 27.

The second example states that the program should stop processing the report if it finds the qualifier "End of R" that exists in the line range starting in column position 1 for a length of 0. Note: length of "0" indicates END OF LINE.

Quick Tip:

The DLM section uses line patterns and qualifiers to represent all line types in a report. The purpose of this section is to give an "identity" to every line. Once the identity of the line has been established, the program can break the line down based on coordinates provided in the DLC section. If the line is identified as one that contains no useful information, it can be ignored.

DLC Section-

The Data Line Coordinate (DLC) section follows the DLM section in the RDF. In the DLC section, the program is given a set of coordinates that allow it to find data in the report. For each line or sequence type, the DLC section tells the program where useful data is located. The coordinates tell the program where each piece of data begins on a line and how long each data field is. Each set of coordinates is also assigned to one of the fields created in the FLD section. This tells the program where to store the data after it is picked up. Here's a look at a completed DLC section:

```
DLC SECTION
[Marker,Group,Seq,Field,Start,Length,Rule,Value,Action]
D001,0,0,2,49,3,,,
D002,0,0,3,41,8,,,
D003,0,0,4,80,19,,,
D004,0,0,5,1,4,=, ,prev
D004,0,0,6,8,24,=, ,prev
D004,0,0,7,34,17,,,
D004,0,0,8,52,24,,,
D004,0,0,9,80,17,,,
D005,0,0,1,1,1,a,T,set
D005,0,0,5,1,4,,,
D005,0,0,6,8,24,,,
D005,0,0,9,103,16,,,
```

(fig. 4)

There are 9 data fields in the DLC section: [Marker, Group, Seq, Field, Start, Length, Rule, Value, and Action]

- **Marker** is either a line marker or sequence marker. It corresponds with the action taken in the DLM section. For instance, a line type may have an **Action of D001** in the DLM section. This action would point to the coordinates that had a **Marker of D001** in the DLC section. Here's an example:

  DLM Section
  27,=,Reporting,0,0,**D001**

  DLC Section
  D001,0,0,2,49,3,,,

- **Group** identifies a subset of coordinates in a marker.
- **Seq** identifies a subset of coordinates in a marker or group.
- **Field** assigns each set of coordinates to a field defined in the FLD section.
- **Start** refers to the starting column position of the data field.

- **Length** tells the program how long the data field is in relation to the start position.
- **Rule** acts as a rule to the action. For example, when the rule "a" (always) is applied, the action on that set of coordinates will **always** take effect. Rules: =, =p, a
- **Value** is used along with certain rules and actions, including "set" and "catv". The value field can be any combination of characters, numbers, or blanks.
- **Action** refers to the action taken on a set of coordinates. There are many actions that provide a developer with control over how data is picked up and stored. Actions include: set, seth, keep, hold, get, getc, getz, cat, and catv.

An explanation of these actions, grouping, sequencing, and use of rules in the DLC section is included in the Design section of this guide.

Now, let's look at a line….

[Marker, Group, Seq, Field, Start, Length, Rule, Value, Action]
D001, 0, 0, 1, 1, 25, , ,

The D001 marker is the link between the line in the DLM section and the set of field coordinates in the DLC section that belongs to it. The piece of data we are picking up starts in column position 1. The data field is 25 characters long. Whatever data is picked up within these coordinates will be stored in Field 1 of the write buffer. The data has no group or sequence and no special rules, actions, or values.

Quick Tip:

The DLC section provides the map to what field data exists on a particular line. It also provides some flexibility for handling the field data through a set of rules and actions.

FLD Section-

We now know that data is picked up using coordinates in the DLC section. But where does it go from there? The job of the DataMagnet is to get and organize data in a neat, columnar format. In the FLD section, we define what I'll call the write buffer because it contains a list of report fields that will be written to a file when triggered. The FLD section gives us control over naming the fields, choosing output order, and formatting the field data. Take a look at a completed FLD section:

```
FLD SECTION
[Field,Output,Type,Order,DataLabel,Label,Action]
1,y,r,1,0,RecType,c
2,y,c,2,0,Currency,
3,n,n,0,0,Exchange Rate,
4,n,c,0,0,,
5,y,c,3,0,Cost Center,
6,y,c,4,0,Description,
7,y,c,5,0,Account,c
8,y,c,6,0,Account Description,c
9,y,n,7,4,Period Activity,c
```

(fig. 5)

There are 7 data fields in the FLD section: [Field, Output, Type, Order, Data Label, Label, and Action]

- **Field** assigns a number to each field. The fields are numbered in order beginning with 1.
- **Output** is the default setting that tells the program whether or not the field will be displayed in the output file. If the field is set to **"y"**, the field will be displayed. If the field is set to **"n"**, it will not be displayed. Note: the user can change this by double clicking the output field in the field list window.
- **Type** sets the format for the data. The field can be set to one of the following.
  r - character data (Text) used for Record Type Field only. A record type field is hidden from the DataMagnet field list window but is written to the output file.
  c - character data (Text)
  d - Date (mm/dd/yy)
  n - 0.00 numeric (2 decimals)
  n0 - 0 numeric (No decimals)
  n1 - 0.0
  n2 - 0.00
  n3 - 0.000

n4 - 0.0000
n5 - 0.00000
n6 - 0.000000
nf - 0.0###### numeric (Float decimals)
w - wraptext (Width 50)
w4 - wraptext (Width 40)
w6 - wraptext (Width 60)
w8 - wraptext (Width 80)
w1 - wraptext (Width 100)

Note: Special Formats above control Excel files only.

- **Order** controls the order in which the fields are displayed. The order does not have to match the field number. Setting the order to **0** (zero) will hide the field from the field list in the DataMagnet. This is commonly used for Data label fields.
- **Data Label** is used to make column headers out of data from other fields. The data label field is set to the number of the field that contains the desired data. For example, if I wanted to label Field 10 with a date I picked up in Field 8, I would set Field 10's data label to 8. If the Data Label is in use (greater than 0), **it will overwrite the value in the Label field.**
- **Label** allows you to name each field. Whatever value is entered in this field will appear as the column header for that set of data.
- **Action** refers to the action that will be taken on the field. In the FLD section, there are two actions: **"c" and "z"**. **"C"** clears the field after writing the data. **"Z"** zeroes the field after writing the data.

Building Data Labels and formatting data are covered in detail in the Design section of this guide.

Let's take a look at a FLD line….

```
FLD SECTION
[Field,Output,Type,Order,DataLabel,Label,Action]
1,y,c,1,0,Entity,
```

In this example, we are looking at a FLD line written for Field 1. Field 1 will be displayed ("y") and its data is character formatted ("c"). It is ordered first for display and does not use a Data Label. The field is labeled "Entity" and there is no action being taken on the field.

Quick Tip:

The FLD section defines columns for data to be written to. In this section, the fields are formatted, labeled, and ordered for display in a columnar format. The FLD section allows a developer to control how the fields are written, how they are labeled, and if they will be displayed. Data formatting offers further control over the output file.

The RID and FLG Sections

The Report Identification (RID) and Flag (FLG) sections are used to identify which Report Description Information to load from a Multiple RDF. These two sections go hand and hand. The program sets flags by scanning for the markers in the RID section. Then it matches the flags with the flag matrix in the FLG section to get the report description collection ID number.

RID Section-

The Report Identification (RID) section is where we define Markers that are used to set a series flags:

```
RID SECTION
[FlagID,Start,Rule,Marker,Action,Length]
F1,1,=,Suppress Zeroes: n,
XX,1,x,Batch,stop,0
```

(fig. 6)

The RID section has 6 data fields [Flag ID, Start, Rule, Marker, Action and Length]

- **Flag ID** for the flag to set. F1,F2 …F16. XX = Dummy.
  The ID must begin with "F" to set a Flag.
  The number is there for reference and has no significance.
  The Flags set in the order of the Markers.
- **Start** refers to the column position that the Marker begins in. If "x" rule where to begin scanning.
- **Rule** determines if the line will be matched to a pattern or to a qualifier.
  = Equal. Match the Marker
  p pattern match.
  x Exists. Scan line for Marker match.
- **Marker** is the qualifier to look for or pattern to match.
- **Action** to perform. **Stop** is current the only action.
- **Length** is the length of the Report line to scan for the "**x**" rule.

In the (fig. 6), Flag 1 (F1) is **set** if the program finds the marker, "Suppress Zeroes: n," in column position 1.

XX is a dummy flag. This line in the screen shot tells the program to stop processing the report if it finds the qualifier, "Batch," in column position 62.

Stop always tells the program to stop processing the report.

FLG Section-

The FLG section tells the program which RDF to use to process the report. Let's take a look:

```
FLG SECTION
[ID,F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14,F15,F16]
01,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
02,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

(fig. 7)

The FLG section has 2 types of data fields: [ID, F1-F16]

- **ID** refers to the RDF Collection ID. "01" represents **RDF collection 1** while "02" represents **RDF collection 2**.
- **F1-F16** is the flag matrix. . If the field is set to "0", the flag is **not set**. If the field is set to "1," the flag is **set**. Each flag represents a marker in the RID section. If a Marker is found the flags is set.

In (fig. 7), we see that Flag 1 (F1) is **set for ID "02" and not set for ID "01"**. Therefore, **if the marker is not found**, RDF collection 01 will be used to process the report and **if the marker is found**, RDF collection 02 will be used to process the report.

Quick Tip-

When a collection ID is found the program will load the corresponding Report Description Collection information from the RDF file.

RLF Section-

The Report Line Format (RLF) section is where output format types and their values are maintained. Note, these settings will override the global settings for the DataMagnet. By default, the global settings are installed to output all records to an Excel spreadsheet and open the file when finished. This might be sufficient for the majority of your reports. However, you might a few reports where you need the data in a delimited text file for importing into another system. If you don't want to change the session settings manually each time for this particular report you can setup the custom format in this section.

There are 2 data fields for the RLF section: [Type, Description]

- **Type** refers to the format type.
  - **HD** Report Header is a special type for the Report Title. (Required) Its description is displayed above the field section window for the RDF identification.

  (All the below types are optional)
  - **OF** Output format. (.xls, .csv, .txt)
  - **QS** Apply text Qualifier switch. (1=yes, 0=no)
  - **TQ** Text qualifier (" ')
  - **DL** Delimiter (| ! ; , tab)
  - **RT** Record Types to report. (All, Details only, Totals only)
  - **ES** Open Excel file Switch. (1=yes, 0=no)

- **Description** is the value of the format type.

Example:

```
RLF SECTION
[Type,Description]
HD,Cost Center Report
OF,.txt
QS,1
TQ,"
DL,!
```

(fig. 8)

In the above example, The RDF was designed for the Cost Center Report. Its going to create a delimited text file with the character field values qualified with double quotes and fields separated by an Exclamation point "!".

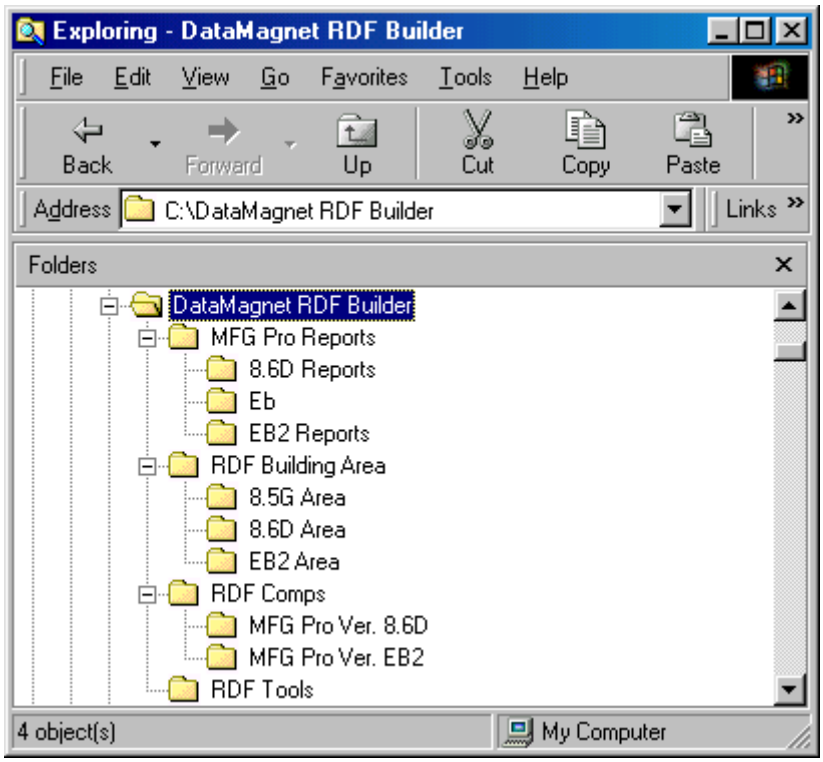
RDF Design

RDF Design-

In this section of the guide, we're going take a look at designing an RDF. In this section, you'll learn how to use the rules and actions for each section in the RDF. We'll also cover grouping and sequencing techniques and logic. You'll learn how to use write switches and how to make "efficient" line patterns.

Creating a Workspace-

Creating Report Description Files requires the creation of an organized workspace on your computer. By creating a separate directory for RDF construction, you can easily access any reports, templates, or other tools that are needed. Here is an example of a directory setup for RDF construction:



The directory, "DataMagnet RDF Builder", has four (4) basic sections –

1. Reports- Sample reports used to write the RDFs are stored here.
2. RDF Building Area- Contains RDFs that are "in progress".
3. RDF Comps- Contains completed RDFs.
4. RDF Tools- Any templates or other tools are stored here.

Within each section, you can create sub-folders to keep related files together. For example, this directory allows the programmer to keep reports and RDFs separate based on their version (EB2,EB,8.6D, or 8.5G). All EB2 reports are kept together, as are 8.6D reports.

A programmer could setup this directory in many different ways. As long as everything is easy to find, the directory can be considered effective. This simple preparation will keep everything organized and will cut down on the total time it takes to write the RDF.

RDF Templates-

It's a good idea to make an RDF template. This allows you to get right to work instead of having to set up a new RDF every time. By template, I mean something that has all the sections and data fields already written out. Having a few example lines for each section won't hurt either. In fact, the FLD section should have 20 to 30 lines with blank labels already written out. This way, you'll only have to fill in the label names and make minor changes when you actually write the RDF.

Designing the DLM-

In the DLM section, we want to give a distinct identity to every line type in the report. We know that we can use data qualifiers or patterns to do this. Whenever possible, we want to use data qualifiers to identify line types. Patterns take longer to process and are more vulnerable to changes in the report layout.
A data qualifier can be any piece of data that always shows up in the same location on the report. Here's a situation where the data qualifier could be used:

Here, we have three "hard-coded" pieces of data: "Project:", "Description:", and "Begin Date:". Any of these pieces of data could be used as data qualifiers. They all represent the same line type. Let's use "Project:" for an example.

"Project:" starts in column position 1. We won't worry about grouping or write switches here. Our DLM line would look like this:

1, =, Project:, 0, 0, D001

This action would then point to a set of coordinates in the DLC section that told the program where the data was located on the line. In this case, we would have coordinates that would pick up project information, the project description, and the begin date.

Now for the patterns.

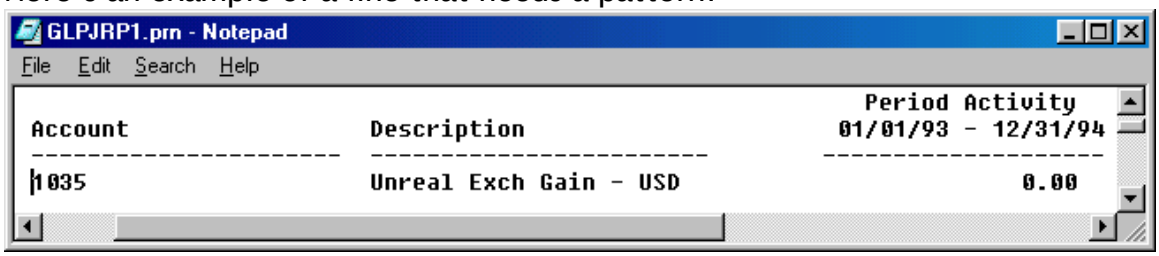
Many times, there is no data qualifier for a line. In these situations, the program has to use a pattern match. The ideal "efficient" pattern is long enough to clearly distinguish the line from all others, but short enough that it won't hurt process time. In other words, you don't want to go overboard with the patterns. Only use what you need to identify the line.

Patterns are created using 6 types of characters:

- **"x"**- tells the program something has to be in that position.
- **"n"**- tells the program a number has to be in that position.
- **"s"**- tells the program something has to be within the range of the "s" characters in the pattern. (Ex. nbooobsssssssssssnnb)
- **"d"**- tells the program that a decimal point has to be within the range of the "d" characters in the pattern. (Ex. nbooobddddddddoob)

- **"b"**- tells the program a blank must be in that position.
- **"o"**- tells the program to ignore that section of the line.
- **Special characters { . , / \ % etc.}** must match character in that position.

Here's an example of a line that needs a pattern:



How would we start writing a pattern for this line? First, look for special characters. In this example, we have a decimal under Period Activity. The decimal starts in column position 62. Next, scan the report to make sure the decimal is always in the same place. Make sure there are no extended decimals in the report. If it appears the decimal is consistent throughout the report, use it. In fact, you can use the numbers around the decimal as well. Here's a possible pattern for this line:

61, p, n.nnb, 0, 0, D002

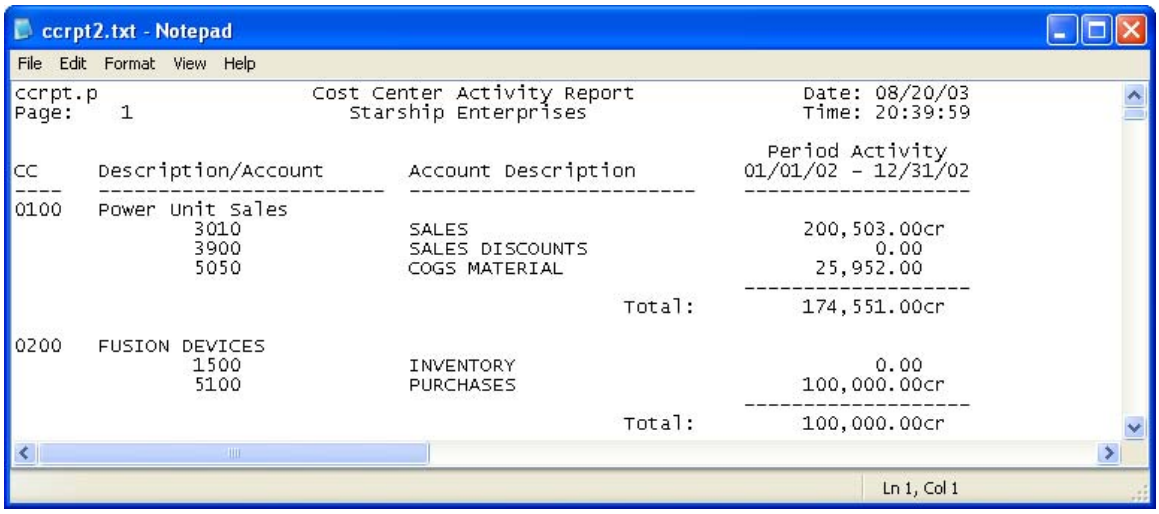
The line will be identified as line type D002 if the program finds a number, then a decimal, then a number, then a number, and then a blank starting in column position 61.

Things to remember when making a pattern:

- Make sure that the pattern only qualifies one line type.
- Make the patterns "efficient".
- Take advantage of special characters.
- Avoid using "x" if you are not 100% sure something will be there.

Write Switches-

The write switches simply trigger a write of the write buffer (FLD Section) to the output file. The write switch field in the DLM section is an integer value. A zero value will not trigger a write. A positive value will trigger a write. The reason this switch is an integer and not a "yes/no" is that we have a need to trigger writes at multiple report levels. Therefore, a write switch can be 1,2,3, etc … where 1, is report level 1 and 2, is report level and so on. For a better understanding, look at the following report example.

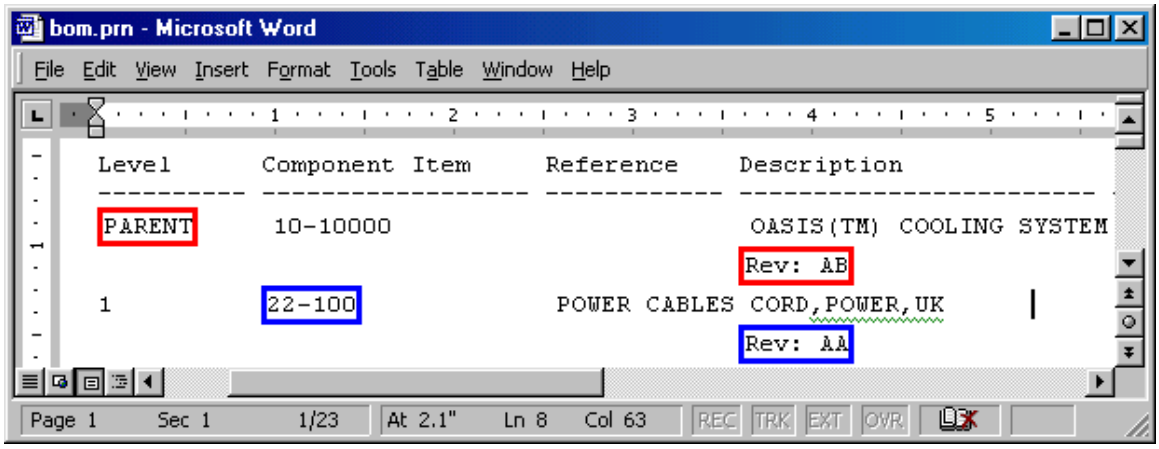


This report has 2 levels. Level one, is the header line (CC and Description). Level two is the detail line (Account, Account Description and Activity). Our write buffer will consist of these 5 fields. The DLM for each line will have a write switch and will look like this.

```
DLM SECTION
[Start,Rule,Marker,Group,Write,Action,Length,SeqGroup]
1,p,nnnnbbb,0,1,D001,,
73,p,n.nn,0,2,D002,,
```

The DataMagnet needs to get the data before it can write it so the first time the header line is found the CC and its Description are stored in the write buffer but not written and write switch #1 is set to "on" to trigger a write the next time a header line is found. This is also true for the detail line. The first time it is found the Account, Account Description and Activity are stored in the write buffer and write switch #2 is set to on. When the next detail line is found the DataMagnet will perform a write before storing the detail data for that line. Therefore, writing the header and first detail line data to the output file. When the next header line is found, the DataMagnet will perform a write of the previous header and its last detail line because its switch (#1) is on. Then it gets the next set of header data and resets the detail line write switch (#2) to off so a write won't be triggered on its first occurrence.

Grouping-
Grouping logic allows the program to assign data to separate fields based on unique line markers. The grouping logic is applied when there are two or more fields of data that cannot be distinguished from one another. Take a look at the following screen shot:



In this case, we have two pieces of data for the "Revision" field. The data in the red box is related to the "Parent Item", while the data in the blue box is related to the "Component Item". The program, however, doesn't know this. If we did not incorporate grouping here, the "Parent Rev" and the "Component Rev" would end up in the same data column.

To avoid this, we can set groups on the Parent line marker and on the Component line marker. Now, when the program finds the "Rev" line in the report, it not only looks in the DLC for coordinates with the corresponding line marker, but for coordinates with the corresponding group as well.
Let's set the first group on the Parent line marker:

1, =, PARENT, **1**, 0, D001

Now, let's set the second group on the Component line marker:

1, p, xooooooooobxooooooooooooooooob, **2**, 0, D002

When the program finds a line that matches "PARENT", the group will be set to 1. The group will be changed to 2 when the program finds a line that matches the pattern. The group does not change for the lines located between these markers. Therefore, the "Rev" line will belong to the group that was set from the line before it.

The Rev Line:
44, =, Rev:, 0, 0, D003

When the program finds a line that matches "REV:" in the report, it will look for coordinates in the DLC section with the same action marker and group. In

this case, coordinates for the Rev data will point to two separate fields depending on the group setting.

```
D003,1,0,14,48,8,,,
D003,2,0,15,48,8,,,
```

- If the group is set to 1 when the data is found, it will be written to field 14 – "**Parent Rev**".
- If the group is set to 2 when the data is found, it will be written to field 15- "**Component Rev**".

Incorporating grouping into the RDF design avoids incorrect placement of similar data. It allows the program to identify data based on a type of line marker located before it in the report.

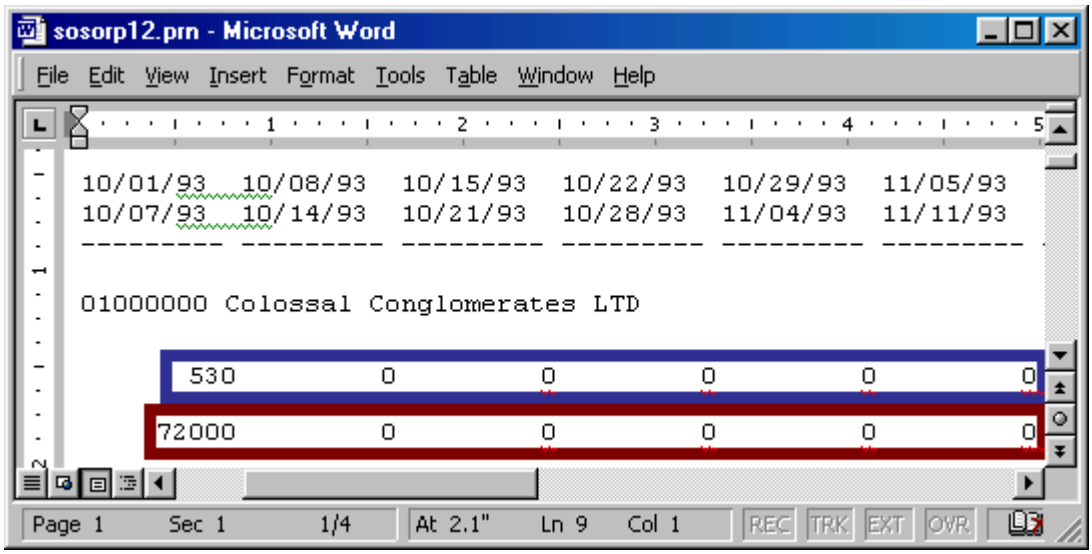
Advanced Grouping Functions

Groups can be both positive and negative integer numbers. Yes, negative. So what's the difference? The difference is when the group is set. Before (Positive) the marker's line data is stored in the write buffer field or After (Negative) the marker's line data is stored in the write buffer field. This adds flexibility however the majority of the time positive markers will achieve the desired result for data capture.

Group Sequencing is active when the "SeqGroup" flag is set "Y" in the DLM line with a group. In this case, the group will increment if the marker was found again and its DLM group matches the last DLM group setting.

Sequencing-

Sequencing logic allows the program to assign data to separate fields based on the order of sequence markers in the report. Sequencing is used in instances where the same line marker occurs multiple times between groups or write switches. Each line marker contains different pieces of data that belong in separate data columns. Here's an example of this situation:



In this situation, we are seeing two sets of values for a series of 6 date ranges. The values in blue are "Quantity" values, while the values in red are "Sales" values. However, there are no differences in the line patterns. There are no distinct qualifiers and the values all start in the same columnar positions. Because this is all information concerning "Colossal Conglomerates", we want all of the data to be written on one line in the output file.

Here's where sequencing logic comes in. Sequencing allows the program to distinguish data based on the order of sequence markers. The first time the program finds this line, the sequence is already set to 1. This tells the program to write the data to the corresponding "Quantity" data fields. The second time it comes across the line, the sequence is set to 2. This tells the program to write the data that it picks up to the proper "Sales" fields. When the program hits a write switch or group marker, the sequence is reset to its highest level – in this case 1. Here's how it's done:

The lines both fit the same pattern:
9, p, nbooooooonb

Now, we make the pattern into a sequence marker by adding a sequence marker code (S001-S999).

The actual sequence marker will look like this:
9, p, nbooooooonb, 0, 0, S001

When the program finds the sequence action marker the first time, it looks for coordinates in the DLC section with the same sequence marker and sequence setting (1). It will find these coordinates:

```
S004,0,1,5,1,9,,,
S004,0,1,9,11,9,,,
S004,0,1,13,21,9,,,
S004,0,1,17,31,9,,,
S004,0,1,21,41,9,,,
S004,0,1,25,51,9,,,
S004,0,1,29,61,9,,,
S004,0,1,33,71,9,,,
S004,0,1,37,81,9,,,
S004,0,1,41,91,9,,,
S004,0,1,45,101,9,,,
S004,0,1,49,111,9,,,
```

The second time the program comes across the sequence marker; the sequence is set to 2. Now it will look for coordinates that have the same sequence marker, but a sequence setting of 2. It will now find these coordinates:

```
S004,0,2,6,1,9,,,
S004,0,2,10,11,9,,,
S004,0,2,14,21,9,,,
S004,0,2,18,31,9,,,
S004,0,2,22,41,9,,,
S004,0,2,26,51,9,,,
S004,0,2,30,61,9,,,
S004,0,2,34,71,9,,,
S004,0,2,38,81,9,,,
S004,0,2,42,91,9,,,
S004,0,2,46,101,9,,,
S004,0,2,50,111,9,,,
```

Notice how changing the sequence settings tells the program to store the data to different data fields - even though it is picking up data from the same locations on identical lines. Sequencing allows the program to identify the data. Because of this, we can control where the data is stored in the write buffer.

DLC, Rules and Actions

[Marker, Group, Seq, Field, Start, Length, Rule, Value, Action]
D001, 0, 0, 1, 1, 25,=, , skip

The Data Line coordinate section consists if 9 fields. The first 6 are basically used to assign a report data element (RDE) to a specific write buffer field. However, in some cases, we want to have more flexibility that just storing data in the field.
For example, we might want to store it for later use, clear the write buffer field based on a value or assign a value to a field. The last 3 fields (Rule, value and Action) provide a means of control for storing data in a write buffer field.

Before we go any further, you need to know that each write buffer field has 2 additional storage areas. One stores the last data value of the field that was written. The other is a temporary storage area (HOLD) for that field.

Rules There are only 3 rules. These are: **=**, **=p**, and **a**. The "=" and "=p" rules are conditional rules for applying an action.
"**=**" Equals, rule checks if the report data element (RDE) is equal to the Value field. If so, one of 2 actions can be performed. These are "skip" and "prev".

"**=**p" Equals Previous, rule checks if the (RDE) is equal to the last data value that was written for the field. If so, there is only one action "set" that can be used.

"**a**" Always, rule performs the Action command listed. All actions are available except prev and skip.

Value is used along with certain rules and actions. The value field can be any combination of characters, numbers, or blanks.

Actions refers to the action taken on a set of coordinates. Actions include:
set Sets the write buffer field (WBF) to the value field.
seth Sets the temporary storage field area (Hold) to the value field.
keep Stores the RDE into the WBE one time and does not over write it.
skip Skips storing the RDE to the WBF.
prev Stores the last value written (previous) in the WBF.
hold Stores the RDE into the temporary storage field area (Hold).
get Gets the value of the HOLD area and stores it into the WBF.
getc Gets the value of the HOLD area and stores it into the WBF. Clear Hold.
getz Gets the value of the HOLD area and stores it into the WBF. Zero Hold.
cat Concatenate the RDE to the value of the WBF.
catv Concatenate the Value field to the value of the WBF.

RDF Template

The following is a RDF template is included to assist with development of a RDF.

To create a RDF use a text editor such as Notepad or Wordpad and save the file as a plain text file with a .rdf extension. Its advised to define your field section (FLD) first. Next, define the line markers (DLM) and finally the line coordinates for the fields (DLC).

```
RDF 0001S progname.p  @Company Name

DIN SECTION
[Type,Description]
DT,Created On:  mm/dd/yy
SV,ssssssss Version:  xxxx
DM,DataMagnet Version:  xxxxx
LS,Licensed To:  Company Name


RID SECTION
[FlagID,Start,Rule,Marker,Action,Length]
F1,01,=,Suppress Zeroes: n,
XX,1,x,Batch,stop,0


FLG SECTION
[ID,F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14,F15,F16]
01,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
02,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0


RDF 01 progname.p

DLM SECTION
[Start,Rule,Marker,Group,Write,Action,Length,SeqGroup]
1,=,progname,0,0,skip
1,=,Page,0,0,skip
80,=,-----,0,0,skip
80,=,Period,0,0,skip
1,x,End of R,0,0,stop,0
1,=,Marker1,0,0,D001
10,=,Marker2,0,0,D002
20,=,Marker3,0,0,D003
30,p,bbxxxxbsssssssssssssssssssssssbbooooooon.nn,0,1,D004


DLC SECTION
[Marker,Group,Seq,Field,Start,Length,Rule,Value,Action]
D001,0,0,1,1,3,,,
D001,0,0,2,1,3,,,
D001,0,0,3,1,3,,,
D001,0,0,4,1,3,,,
D001,0,0,5,1,3,,,
```

```
D002,0,0,6,1,8,,,
D002,0,0,7,1,8,,,
D002,0,0,8,1,8,,,
D002,0,0,9,1,8,,,
D002,0,0,10,1,8,,,

D003,0,0,11,1,19,,,
D003,0,0,12,1,19,,,
D003,0,0,13,1,19,,,
D003,0,0,14,1,19,,,

D004,0,0,15,1,4,,,
D004,0,0,16,1,24,,,
D004,0,0,17,1,10,,,
D004,0,0,18,1,10,,,
D004,0,0,19,1,10,,,
D004,0,0,20,1,10,,,
D004,0,0,21,1,10,,,
D004,0,0,22,1,10,,,
D004,0,0,23,1,10,,,
D004,0,0,24,1,10,,,
D004,0,0,25,1,10,,,
D004,0,0,26,1,10,,,
D004,0,0,27,1,10,,,
D004,0,0,28,1,10,,,
D004,0,0,29,1,10,,,
D004,0,0,30,1,10,,,



FLD SECTION
[Field,Output,Type,Order,DataLabel,Label,Action]
1,y,c,1,0,label,
2,y,c,2,0,label,
3,y,c,3,0,label,
4,y,c,4,0,label,
5,y,c,5,0,label,
6,y,c,6,0,label,
7,y,c,7,0,label,
8,y,c,8,0,label,
9,y,c,9,0,label,
10,y,c,10,0,label,
11,y,c,11,0,label,
12,y,c,12,0,label,
13,y,c,13,0,label,
14,y,c,14,0,label,
15,y,c,15,0,label,
16,y,c,16,0,label,
17,y,c,17,0,label,
18,y,c,18,0,label,
19,y,c,19,0,label,
20,y,c,20,0,label,
21,y,c,21,0,label,
22,y,c,22,0,label,
23,y,c,23,0,label,
24,y,c,24,0,label,
```

```
25,y,c,25,0,label,
26,y,c,26,0,label,
27,y,c,27,0,label,
28,y,c,28,0,label,
29,y,c,29,0,label,
30,y,c,30,0,label,


RLF SECTION
[Type,Description]
HD,Title Report


RDF 02 progname.p

DLM SECTION

DLC SECTION

FLD SECTION

RLF SECTION
```