

ResourceMiner User Guide

**version 2.2
2009-02-09**

**Copyright (c) 2001-2009 by Gruvan Application Mining, Sundsvall, Sweden.
All rights reserved.**

Table of Content

<i>Getting started</i>	3
Create a database	3
Make ResourceMiner aware of a database	3
See which database is connected	3
Change database	3
Load source into a database	4
Check the quality of loaded source	4
ParseErrors	4
Duplicate Source	4
Missing Source	4
Check the need for middleware settings	5
<i>How to use ResourceMiner</i>	6
General	6
Error analysis	7
Change Analysis	9
Test case analysis	11
Harvest business rules	13
Understand unknown systems	15
Common use cases	17
Search with Query for Statements	17
How is the system organized?	18
How is the application organized?.....	18
How is the program organized?.....	18
What is the program part of?	18
What is the object (class/form/screen/record/table/datafile) part of?	18
What is the entry (method/function/process/procedure/section) part of?	18
From where is the object (class/form/screen/record/table/datafile) used?	19
From where is the entry (method/function/process/procedure/section) called?.....	19
Which other objects does the object (class/form/screen/record/table/datafile) use?	19
Which other entries does the entry (method/function/process/procedure/section) call?.....	19
Show the callchain startingpoints	19
Show the callchain starting at the entry	19
Read code in execution order starting at the entry.....	19
<i>Middleware for Dynamic SQL</i>	20
<i>Support</i>	22



Getting started

Create a database

At installation ResourceMiner has only one database; RM_FirstDatabase which is of type MS Access. RM_FirstDatabase is empty and ready for load, therefore it is not necessary to create a database to get started.

A new empty database in SQL Server, DB2, Oracle or other database system is created by adjusting and run the DDL-script **RM_Empty20_DDL.txt**. The database and scheme (if used) can have any names, tables and indexes must not be changed.

Make ResourceMiner aware of a database

ResourceMiner can only connect to databases it is aware of, make databases known in the mainmenu choice **Tools\Settings\Databases**:

The list contains the known databases, only one database can be *Default*, ie be the database that is connected when ResourceMiner starts (if starting ResourceMiner by opening a rmv- or rmq-document, the document itself contains what database to connect to, which can be another database than the default).

The button *Add* adds a database to the list:

1. Set database name, can be any name that says something about it's content, for example the system name or the company name if the source database contains all or several systems for a company.
2. Choose database type. Read the blue text which change with selected database type.
3. Depending on database type, create the dbfile by a click on the 'Create'-button or set the connection string. För InterBase (which is the same as FireBird) point out the IB-file.
4. Save and Close.

Tip: Do not set the database to default until tested that the database connection is working.
Test connect by *File/Source Databases*.

The button *Delete* removes a database from the list (the physical database is not removed, only ResourceMiner knowledge about it is affected).

See which database is connected

View- and Query-document shows the connected database in the caption of its window, for example *View [1] on MyDB*.

Change database

The mainmenu choice **File\Source Databases** give the possibility to connect the current form to another source database and will show when source databases was loaded last time. Connection to source database is inherited from current form into new forms when doing 'New...' or 'Show in...' functions. The database connection is stored with a saved document.

Load source into a database

This instruction is the shortest and simplest way of loading source files to get started, read more about the Load function in the Manual.

1. Make sure the source files are available as readable text files in a folder or tree of folders.
2. Choose **Tools/Settings/Languages** at the mainmenu and check that the source files is matching the file extensions given for the language definitions of interest. Change the file extension settings if needed.
3. Choose **Tools/Load** at the mainmenu.
4. Add one or several folders as '**From Folder**', by clicking the button *Add* and point out the folders, choose or type System, Computer and Application names, any names will do, and click the button *Save*.
5. Click the button *Load*. Loading the database can take some time depending on the amount of source code and the performance of the computer.

If the load is interrupted, for example by shutting down the computer or killing the application in task manager, it is possible to restart the load later on. ResourceMiner know where it stopped and will start from that point next time Load is started.

Secondary loads (keeping the database up to date with the source files) only involves step 3 and 5 in this instruction. Only changes will be handled, the load time will be much shorter.

For secondary loads it is possible to load automatically by Scheduled Task, see example script **BackgroundLoad_RMDBs.bat** that comes with the installation and contains comments on what it does.

Check the quality of loaded source

To be sure of a correct picture of the applications inner structure, do the following checks:

ParseErrors

Choose **File/Open/Query SQL Report** at the mainmenu, doubleclick the file *CheckParseErrors.rmq* and click the *Search* button. It should be no hits in the result.

If parse errors exists, check that the source files extension is matching the language definitions file extension and check that the source file can be compiled in its developer environment. Make necessary changes and load again.

If parse errors still exists, contact ResourceMiner support.

Duplicate Source

Duplicate source (ie copies of same source with different file names) can give a wrong picture of the applications inner structure, be sure no duplicate source has been loaded.

Choose **File/Open/Query SQL Report** at the mainmenu, doubleclick the file *CheckDuplicateObjectNames.rmq* and click the *Search* button.

If duplicate objectnames exists, be sure they are all used. Remove not used copies of the source from its folder and load again with the option *Rebuild all Dependencies*.

Missing Source

Missing source will give a wrong picture of the applications inner structure, be sure all needed source is loaded.

Choose **File/Open/Query SQL Report** at the mainmenu, doubleclick the file *Distinct Missing Objects.rmq* and click the *Search* button.

If missing objects exists, be sure they are not part of the application source. 3:rd party software or platform components is OK to be missing. Use the report *Missing Objects.rmq* for better understanding of how and where the missing objects are used.

Add missing source files to the load folders and load again with the option *Rebuild all Dependencies*.

If the missing objects are already loaded, contact ResourceMiner support.

Check the need for middleware settings

ResourceMiner finds and resolves all static dependencies within the source code, ie includes, calls, pointers etc.

There is also another type of dependencies one should be aware about; dynamic dependencies where a call and it's parameters contains the dependency, normally in a string parameter. This is quite common in modern programming languages but can exist in mainframe languages as well. The called function or program that takes the parameter is a middleware. Middlewares are often 3:rd party products or components and its source code is not available or part of the own source. A middleware makes a call or connection to the System/Subsystem/Object/Entry given as parameters.

To be able to get a correct picture of the applications inner structure within and/or between systems one needs to find and define the missing middleware links. The missing links can be found using ResourceMiner to see what objects are not used at all and then search the source for their names. If the name exists in the code, verify if used as parameters to middleware functions of any kind.

Choose **File/Open/Query SQL Report** at the mainmenu, doubleclick the file *Not used Objects.rmq* and click the *Search* button.

For each object in the result, use Query for Statements (click the tab next to Query SQL Report) and search for:

Any Executable Word equals <objectname>

Verify if <objectname> is used in parameters to middleware functions of any kind and if so, define a middleware to make the connection. See the Manual about how to define a middleware connection.

In some middleware cases, only EntryName is given in the middleware parameters, the objects name is known by systems settings outside the source. The report *Not used Entrys.rmq* can help to find such entries.

See section Middleware for Dynamic SQL below for a setting example for the VB.Net PESO Application that can be downloaded at www.resourceminer.nu/Downloads/VB_DotNet_PESO.zip.



How to use ResourceMiner

The user takes control over applications and its source when loading, checking the source and defining middlewares. See the section *Understand unknown systems* below to learn more about the loaded source.

General

At the least ResourceMiner is an advanced search tool for application source, more intuitive than GREP-like tools. With good database performance it is also much faster than search tools for the file system because the source has been indexed. ResourceMiner 2.2 has a **Wizard** making it very easy to access! See ResourceMiner Manual.

At the best ResourceMiner helps to solve daily problems faster and more accurate than before. That however, requires some new methods of working which may take some time to get used to.

Use ResourceMiner with this principles in mind:

1. Do not look upon the application as a set of source files, instead look at it as a set of callchains that starts at user actions (Program start and Events).

A callchain is like a tree with a root (startingpoint) with branches of code sequences that executes sequentially branch by branch. At each Call, the callchain branches into another Entry of code sequences.

In which source file things happens is not important until the analysis is done and it is time to make changes in the source.

2. Think of variables and fields in datafiles, tables and forms as datacarriers. The data they carry is business information or technical information about the platform. Data is moving from one datacarrier to another in flows.

Common business dataflows goes from database/file up to forms and back down to database/file.

Dataflows happens within callchains, both inwards (towards database/file) and outwards (towards form), the code execution is always done in one direction though (forward).

A dataflow consists normally of only a few lines of code buried within the callchain.

3. Which code in a callchain that is actually executed, and therefore which dataflow that actually happens, depends on which rules on the way that is fulfilled by the incoming data.

The rules are comparisons like if-, while, och case-statements. Sometimes the callchain needs to be followed considering rules and incoming data, sometimes the answer is clear without considering the rules.

The goal for an analysis is to identify one or several callchains and within them do one or more of:

- Understand what is happening
- Find errors in the code
- Find places where code should be changed
- Find other callchains or programobjects that is impacted by a change
- Understand and decide how to test the changes
- Find and describe business rules
- Make graphics of the inner structure to share with others

In many cases there will be need for examine data in databases or files. ResourceMiner is not a tool for examine data, use database management or file system tools for that.

The following sections show some tutorial examples.

Error analysis

When a user reports an error it should be known (1) which form/program the user worked with (2) what the user did and (3) the symptom – error message or other unexpected data output – that the user reacted on.

The user information given should be enough to find the startingpoint for the callchain where the error or unexpected behavior happened. Search the callchain for the error message or where the output field that had the unexpected data is assigned. Examine the rules and if needed other dataflows involved in the rules.

This example is made on the VB.Net PESO Application that can be downloaded at

www.resourceminer.nu/Downloads/VB_DotNet_PESO.zip

A user has reported that a dialogbox 'Data Not Found !' appeared when searching for a company 'Kodak' she know exists in the database. The search was made in the form named 'Employers Information'. Why?

Solution in ResourceMiner:

1. Finding the form can be done looking at the forms in the developer environment or use ResourceMiner:

In mainmenu: **File/New/Query for Statements**

Search for 'Word in String equals Employers' (to find 'Employers Information')

Only a few hits is found and it is quite obvious that the form source is named Employers because the row **Me.Label1.Text = "Employers Information"** is set in the Employers InitializeComponent entry.

2. Find the error message or the output field with unexpected data. In this case the error message:

Search for 'Word in String equals Data and Word in String equals Not'

Three hits is found when showing dialogbox with 'Data Not Found !'

Which of them did the user get?

3. Find the startingpoint for the callchain for each row in the result to decide if the hit is possible:

Leftclick on the row and choose *Show in New View*

Set the View controlpanel to Dependencies-BottomUp and click *Refresh*

Expand (click the + sign) as far as possible. In this case it is not possible to expand because the hits are all in startingpoints. One startingpoint is in another object named List, it can not be the users error message.

Now there are two possible error messages in the entry TextBox10_KeyPress which also is the startingpoint.

4. Understand which error message and why by searching the callchain from the top:

Set the View controlpanel to Dependencies-TopDown, check *Show Entrys Sequentially* and click *Refresh*.

Select the entry TextBox10_KeyPress (click on it), rightclick and choose *Expand All* in the popupmenu. The complete callchain will be showed.

Select the complete callchain by hitting Ctrl+A, rightclick and choose *Query Track for Statements* in the popupmenu. A new Query form opens up.



5. Search the callchain for 'Word in String equals Data and Word in String equals Not' (same as in action 2 above)

This time only two hits is found when showing dialogbox with 'Data Not Found !'

Check *Include Rules* to see the rules that applies to the two hits.

It is clear that the state of RadioButton1 and RadioButton2 decides which of the error messages to get.

What are those radiobuttons?

6. It is common that new questions being raised during the analysis. Find the answers to get further:

Switch to the other Query Window (hit Ctrl-Tab or select it in the Window-menu) to search all source.

Search for 'Any Executable Word begins with RadioButton In Objects Name equals Employers' to find out more about these radiobuttons.

When examining the result it is clear that RadioButton1 is about Job Vacancy and RadioButton2 is about Company Name. The user said that she was searching for a company 'Kodak' so now we know which of the errors she got (supposing she did click RadioButton2 prior her search, otherwise she should try that).

7. Switch back to the callchain Query (hit Ctrl-Tab or select it in the Window-menu).

Rightclick the row with the error message we now know appeared and choose *Show Rows Around* in the popumenu. From the code it is obvious that an exception must been raised in the called entry `data_load2`, so next step is to look into that entry.

Change the Query to 'Any Executable Word equals `data_load2`', click *Search*, rightclick the row that declares the entry `data_load2` and choose *Show Rows Around* in the popumenu.

From the code there is a few possible explanations to the error:

- a) A problem with the database connection
 - b) A problem with the datareader component
 - c) The data did not exist after all (wrong database connected?)
 - d) The user did not click RadioButton2, searching for Job Vacancies instead
 - e) The user misspelled the company name (never trust a user)
8. With current knowledge about the problem and the applications inner workings according to the problem it is possible to go back to the user and ask some more questions to exclude each of the possible explanations.

The source involved in this example is all located in the same source file and the entry code sequences are short and simple. Therefore it might go faster to just open the source file and read the code to understand the possible error explanations. In larger applications callchains runs over several source files and/or have long entry code sequences filled with rules. These are the applications that ResourceMiner can save a lot of time on.

This way of doing error analysis in ResourceMiner is generic and repeatable to all kind of problems. Make it a habit to identify and search callchains instead of open source files and read code. It will save you time in the long run.

Change Analysis

A request to change an application normally comes from the customers super users describing the changes (1) in business terms and may be (2) which forms involved and/or (3) which tables or files to be changed.

A change analysis will probably raise new questions that only the customer or super user can answer. A well performed change analysis will raise questions of importance and help the customer making decisions. These people are often busy and will appreciate not be bothered with too much simple questions.

This example is made on the VB.Net PESO Application that can be downloaded at

www.resourceminer.nu/Downloads/VB_DotNet_PESO.zip

Note: To get the same result in step 3 in this example, add middleware settings for dynamic SQL (see section Middleware for Dynamic SQL below).

The change request is to add a new field Address2 to the Company information. It should be updated and showed just like Address. Before implementing the change the customer wants to know how much work it takes to decide if it is affordable.

Solution in ResourceMiner:

1. Finding the database tables can be done using the database manager tool or use ResourceMiner:

In View: Distribution-TopDown-View By Object, Filter 'Objects Type equals TABLE', click *Refresh*. These are the database tables in PESO.

Expand the Company table (click the + sign), all fields are shown. Address is there, 50 chars long.

Our first work would be to add Address2, 50 chars long.

2. Create a WorkList by choosing File/New/WorkList in the mainmenu. Excel opens with a WorkList template.

Switch window to get back to ResourceMiner, select (click) the Company Table or the Address field, rightclick and choose *Add to WorkList as* in the popupmenu.

Fill 'Add Address2 50 chars' in the To do textbox and click *Save*. The first work is added to WorkList.

3. Find out what more that is affected by this change, start at object level:

Select (click) the Company Table, rightclick and choose *Show In New View* in the popupmenu.

Set the new View to Dependencies-BottomUp and click *Refresh*. Expand the Company table (click the + sign), all objects that uses the table will be showed. In this case the forms Employers, List and Register.

To know how they use the Company table, select both the table and one other object, rightclick and choose *Show Dependent Rows* in the popupmenu. The dependent rows are SQL-statements showing that:

The Employers form does four different SELECT, one INSERT, one UPDATE and one DELETE.

The List form does two different SELECT.

The Register form does one SELECT.

With some experience of changing this application, above information is probably enough to estimate how much work the table change will take, which is what the customer need to know to make the decision

about doing it or not. Going further in the change analysis may raise questions of importance and help the customer making even better decisions.

4. Find out at a more detailed level what need to be changed by finding the involved SQL-statements and if needed, follow the dataflow within the callchains involved:

In mainmenu: **File/New/Query for Statements** and search for 'Any Executable Word equals FROM or Any Executable Word equals INSERT or Any Executable Word equals UPDATE and Any Executable Word equals COMPANY' to find all SELECT-, DELETE-, INSERT- and UPDATE-statements for table Company.

For each hit in the resultlist, ask the question: Is change needed to handle the extra Address2 field?

There is 15 hits in this example, only the first five will be covered:

- 1) DELETE * FROM Company ... in Employers.Button6_Click will delete the entire record and Address2 would be deleted too. No work needed.
- 2) MsgBox("Please insert Company... is not an SQL-statement, the statement just happened to have the same words. Ignore this hit.
- 3) INSERT INTO Company(Company, Telephone, Address, ... need to be changed so that Address2 also gets updated. Select the row, rightclick and choose *Add to WorkList as* in the popupmenu. Fill 'Include Address2 in statement' in the To do textbox and click *Save*. The second work is added to WorkList.

Address2 must come from user input just as Address does. How does Address dataflow looks like?

First try rightclick and choose *Show Rows Around*. In this case it is obvious that Address comes directly from TextBox9. A new Textbox for Address2 is needed. Close the window. Rightclick the row again and choose *Add to WorkList as* in the popupmenu. Fill 'Add Textbox for Address2' in the To do textbox and click *Save*.

- 4) UPDATE Company SET...
Same as for INSERT above.
- 5) SELECT * FROM Company ... in Employers.ComboBox1_SelectedIndexChanged will read all fields including the new Address2.

Address2 must be showed in the form just as Address does. How does Address dataflow looks like?

First try rightclick and choose *Show Rows Around*. In this case it is not obvious how Address comes to the form, search the callchain to examine the dataflow:

Rightclick the row again and choose *Show in New View*. Set the View controlpanel to Dependencies-BottomUp and click *Refresh*. Expand (click the + sign) as far as possible. In this case it is not possible to expand because the hit is in a startingpoint. Set the View controlpanel to Dependencies-TopDown, check Show Entrys Sequentially and click *Refresh*. Select the entry ComboBox1_SelectedIndexChanged, rightclick and choose *Expand All* in the popupmenu. The complete callchain will be showed. Select the complete callchain by hitting Ctrl+A, rightclick and choose *Query Track for Statements* in the popupmenu. A new Query form opens up.

Search for 'Any Executable Word equals ADDRESS'. IF a hit like ADDRESS = ADR1 had been found, trace the dataflow by appending OR 'Any Executable Word equals ADR1' and so on until the dataflow is fully understood.

In this case the dataflow is TextBox9.Text = objreader("Address") which is how Address comes to the

form. Address2 should have a similar line of code to get to its TextBox on the form. Rightclick the row and choose *Add to WorkList as* in the popupmenu. Fill 'Add similar for Address2' in the To do textbox and click *Save*.

After taking change decisions on all 15 hits, there is a complete WorkList. Switch to Excel, choose *Save As* and save the WorkList for future use.

5. A more exact estimation can now be delivered to the customer. Probably the details also raise questions of importance that could help the customer making better decisions. Such a question could be: Is it necessary to show Address2 in the List-form?

Some time later when it is time to do the changes in the code, the WorkList is already done down to lines of code to change. There is no need to do the analysis again.

Open the WorkList (from Excel will do), select a row and click the button 'Open selected WorkListRow in Editor' to get to the place to change.

This way of doing change analysis in ResourceMiner is generic and repeatable to all kind of changes. Make it a habit to identify lines of code to change as far as possible before editing any source files. It will save you time in the long run.

Test case analysis

It is not unusual that system developers have limited skills on how to test applications. Finding places in source to change can be far more easy than knowing which test data or user input to use to run that piece of code. With ResourceMiner it is possible to see rules that must be fulfilled to get to the lines of code to test.

This example is made on the VB.Net PESO Application that can be downloaded at

www.resourceminer.nu/Downloads/VB_DotNet_PESO.zip

In the change analysis example above, the new field Address2 is added to same dataflows as Address. All changes should be tested and one of the first test cases would be to insert a new Company record. The user input data must pass some rules in order to be successful.

Solution in ResourceMiner:

1. Find the statement to test:

In mainmenu: **File/New/Query for Statements** and search for 'Any Executable Word equals INSERT and Any Executable Word equals COMPANYY' to find the insert statement.

2. Find the startingpoint for the statement:

Leftclick on the row and choose *Show in New View*.

Set the View controlpanel to Dependencies-BottomUp and click *Refresh*.

Expand (click the + sign) as far as possible. In this case it is not possible to expand because the hit exists in a startingpoint. The form to use is named Employers because the entry exists in that object.

3. See the rules by searching the callchain from the top:

Set the View controlpanel to Dependencies-TopDown, check *Show Entrys Sequentially* and click *Refresh*.

Select the entry `Button8_Click`, rightclick and choose *Expand All* in the popupmenu. The complete callchain will be showed.

Select the complete callchain by hitting `Ctrl+A`, rightclick and choose *Query Track for Statements* in the popupmenu. A new Query form opens up.

In this new Query, search for ‘Any Executable Word equals INSERT and Any Executable Word equals COMPANY’ to find the insert statement.

Check *Include Rules* to see the rules that applies to the hit. The rules are marked with green color.

4. Understand the rules by rightclick and choose *Show Rows Around* in the popupmenu. From the code it is quite obvious what the rules are, because the error messages tells what business data and rule to fulfill.

The last rule, if `Flag = 1` must also be passed. This rule is not obvious how to fulfill.

5. Change the Query to ‘Any Executable Word equals FLAG’, uncheck the *Include Rules*, searching for how flag is handled. Flag is not getting any value in the callchain, it must be set elsewhere.
6. Search more source for Flag by choosing **File/New/Query for Statements** and search for ‘Any Executable Word equals FLAG in Objects Name equals Employers’. (Employers is the form where the startingpoint exists).

The result shows the rules we already know about in the entry `Button8_Click`, but also the statements where Flag is set; `Button5_Click` and `Button7_Click`. `Button5_Click` will give `Flag = 1` which should be fulfilled.

To know which button to click in the form, change the Query to ‘Any Executable Word equals Button5’. The result shows that the button should have the tooltip ‘Add’ on it. (`Button8` which should be clicked to execute the insert statement is named `Save`).

The source involved in this example is all located in the same source file and the entry code sequences are short and simple. Therefore it might go faster to just open the source file and read the code to understand the rules that must be fulfilled. In larger applications callchains runs over several source files and/or have long entry code sequences filled with rules. These are the applications that ResourceMiner can save a lot of time on.

Harvest business rules

It is quite common that system developers get questions from their customers or super users about how a certain business term is handled or how a business function handles its task within the application. It is possible to harvest whole business processes from the source to get requirements for a new system.

This example is made on the small IBM Mainframe Application that can be downloaded at

www.resourceminer.nu/Downloads/IBM_MainFrame.zip

There are two different methods of harvesting business rules; a shorter and less accurate method that can be used to estimate the work needed to build a new system and a more detailed method to harvest complete rules.

Both methods begin with two common tasks:

1. Identify startingpoints to harvest from the application.

Choose **File/Open/Query SQL Report** at the mainmenu, doubleclick the file *CallChain Startpoints.rmq* and click the *Search* button. It should be only one hit in this example, in a real situation it can be hundreds of hits.

Study each starting point and decide 1) if it is part of the scope of harvesting which might be known by searching the callchain for information and 2) if in use which probably needs to be answered from people with system knowledge.

Make a list of startingpoints to be harvested which is the scope of work.

Another way of identifying the scope of work can be to start from database tables or files used within the scope. The CallChain StartPoint query can be adjusted to find only callchains containing those objects.

2. Harvest input and output data for each startingpoint of interest:

Set the View controlpanel to Dependencies-TopDown-Object-Filter Objects Name = <startingpoint> and click *Refresh*. Select the startingpoint object, rightclick and choose *Expand All* in the popupmenu. The complete chain of use at objectlevel will be showed.

For each file, or database table, use *Show Dependent Rows* to see if it is used for input, output or both. If screens are used, document them.

Use Word or Excel to document the information about each startingpoint.

The short and less accurate method requires that the application has good enough comments on what is happening in the code:

3. Search the complete callchain for comments:

Set the View controlpanel to Dependencies-TopDown-Entry-Filter Objects Name = <startingpoint>, check *Show Entries Sequentially* and click *Refresh*. Select the startingpoint object, rightclick and choose *Expand All* in the popupmenu.

Select the complete callchain by hitting Ctrl+A, rightclick and choose *Query Track for Statements* in the popupmenu. A new Query form opens up.

4. Search the callchain for 'Word in Comment equals *' and copy&paste all comments that say something useful about what is happening to the Word- or Excel document for the callchain.

If rules are of interest, check *Include Rules*, the rules apply to comments as if they were executable statements.

5. For each comment of interest, see how much code involved (use *Show Rows Around*) and rate the complexity for example 1 – 3, use this information for estimation calculations.

Be aware that the same comments and rules can occur in several callchains, because they share source (includes or subroutines) or because code is duplicated. Harvest duplicates only once.

End of the short method, it is possible to go back and harvest more detailed information at a later time.

The detailed method is based on output fields and how data gets there:

3. Search the object use chain for output fields in database tables, files and screens.

Set the View controlpanel to Dependencies-TopDown-Object-Filter Objects Name = <startingpoint>, and click *Refresh*. Select the startingpoint object, rightclick and choose *Expand All* in the popupmenu.

Select the complete object use chain by hitting Ctrl+A, rightclick and choose *Query Track for Statements* in the popupmenu. A new Query form opens up.

For database tables: 'Any Executable Word equals INSERT or UPDATE'

For Files: 'Any Executable Word equals WRITE'. Extend the Query with
or Any Executable Word equals <record name>
or <filedescriptor name> to match the outputfields with its file.

For Screens: 'Any Executable Word equals MAP'. Note the MAP commarea name.
Then 'Variable Name (assignto) begins with <MAP commarea name>
or use whatever Query that will find where those variables is assigned with value.

The task here is to document all output fields that are assigned with a value at any time, but each field need only to exist in the list once. The list can be long. May be only certain business terms is of interest.

In this example, the application only has file output. 'Any Executable Word equals WRITE or Any Executable Word equals OUTFILE' will find the output statements. *Show Rows Around* is needed on the row FD OUTFILE to see the record definition. The output fields are OUT-FILE-FIELD1 – 4.

4. For each output field of interest in the list, search the callchain for places where it is assigned.

Set the View controlpanel to Dependencies-TopDown-Entry-Filter Entrys Name = <startingpoint>, check *Show Entrys Sequentially* and click *Refresh*. Select the startingpoint entry, rightclick and choose *Expand All* in the popupmenu.

Select the complete callchain chain by hitting Ctrl+A, rightclick and choose *Query Track for Statements* in the popupmenu. A new Query form opens up.

In this example search for 'Variable Name (assignto) equals OUT-FILE-FIELD1'. The two statements found wraps over more than one row. Use *Show Rows Around* to see the whole statements. Data comes in both cases from IN-FILE-FIELD1. Check *Include Rules*, copy&paste or describe the rules for this part of the dataflow.

Search dataflow backwards, change the query to 'Variable Name (assignto) equals IN-FILE-FIELD1'. No hits this time, that was the end of this dataflow and its rules.



Harvesting business rules is a very repetitive task. When done in large scale it can be more or less automated with macros or programs. ResourceMiner databases are open and can be accessed by own written programs.

Understand unknown systems

The goal here is to manage the unknown system enough to be able to correct errors, develop or do any kind of work within its source. The first meeting with an unknown system is often to read documentation, look at forms, use the system and in the best of worlds get a good education from experts that will be around for a long time to answer tons of questions.

If the experts are not around and the documentation on the systems inner workings is vague or not updated, ResourceMiner can help. Go through the tasks below, the tasks need not to be done in any particular order, not even all tasks need to be done, it is up to the user when enough knowledge has been achieved.

When the user feel enough safe about the systems inner structure, enough to not disturb the thoughts on details, this activity is done.

1. Load source files and check the load quality, see section Get Started above.
 2. System parts and naming conventions:

View – Distribution – TopDown – View by System: Drill down to see the parts.
What does the system consists of?
Size? (number of object, lines of code, number of x)
Understandable naming conventions?
Well commented code?
 3. If appropriate, reload the system so that Client/Server/Common/Subsystems/Applications and Database is more divided at System/Computer/Application-level.
 4. Does middleware exists? This question should have been answered in step 1 above and if so, define middlewares and reload. Use the Query SQL Report *CallChain Startpoints.rmq* and look for unusual startingpoints. Normal startingpoints are Event (OnClick, OnItemChange etc) and programstart or job (in Mainframe applications). Use *Query for Statements* to search for unusual startingpoint names. If the startingpoint name is found, it might be a call of some sort.
 5. Study the inner structure:
 - a) Use the Query SQL Report *Most central objects.rmq* to find 'the center' of the system.
What do they do?
 - b) Study the most central objects in View – Dependencies, both TopDown and BottomUp, look at DependentRows to see how interaction is performed.
 - c) Use the Query SQL Report *Most central entrys.rmq*.
What do the central ones do? Is there many not used at all? Lot of dead code?
 - d) Use the Query SQL Report *CallChain Startpoints.rmq*.
Which are at 'the center'? What do they do?
 - e) Create some diagrams of the most important callchains.
 - f) Read through callchains with View – Dependencies - TopDown – Code.
What code patterns are used in the system? When familiar = feel safe and confident
 - g) Complex areas? Note or try to understand.
 - h) Simple and straightforward areas? Note and feel safe
-



6. Increase the understanding for callchains and their order of use:

Where does the application start?
How reach other callchains?
 7. Is naming conventions clear for Objects, Entries and Variables? What can be known from the names?
 8. Increase the understanding for the use of components, platform services and integrated applications or 3:rd party software:
 - a) Which componenets from the developer environment (API:s) is used?
 - b) Which platform services (OS) is used?
 - c) Which 3:rd party software is used?
 - d) Integration with other inhouse applications?
 9. Increase the understanding for the informationen used:
 - a) What tables and files is used? Most used? What business terms do they hold?
 - b) If OO-language, which objects carries data and which objects controls their flow?
 - c) Is there tables/files that has data that controls the applications behaviour? How?
 10. Increase the understanding for the business terms and their relations:
 - a) What variables is used in the most important callchains?
 - b) Which of them carries business data? Naming conventions?
 - c) List all variables handled with Query SQL Report:
select distinct(AMWord) where AMWordType = 1011
 - d) What business processes does the system mirror?
 - e) How does business data relate to each other? A lot of rules? Complex business?
 - f) What parts do not handle business terms? How is the technical platform handled?
 11. Search callchains for comments only, include rules. Does it make sense?
 12. The ultimate test is to challenge the system with an error or change analysis.
Try a real errand.
-

Common use cases

Search with Query for Statements

In ResourceMiner databases the source code is divided into words. Characters like () [] + - etc are also words. All words is marked with their type like this:

- Comment
- String
- Variable **where they are declared (ie DIM A as Integer)**
- Variable **where they identify another object (ie B.C = 5 parses B to an identifier)**
- Variable **where there data is assigned or read (ie D = 5)**
- Hardcoded value (numbers, not text in strings)
- Constant
- Assign which is a keyword for giving a variable data (=, :=, MOVE etc)
- Compute which is keywords like + - * / etc.
- Compare which is keywords like < > = etc (the word = can be assign or compare depending on context)

To find this statement: if foo > bar
Search for: Variables (use) equals foo And
 Compares

To find this statement: if foo > 10
Search for: Hardcoded Value equals 10 And
 Compares

To find this statement: if Left(table, 11) = 'four chairs'
Search for: Word in String contains chairs And
 Compares

As default the search wordtype is **Any Executable Word** which may result in too many hits in the result. To narrow the search either 1) add another search condition for the statements to find (And Any Executable Word equals xxx) or 2) change wordtype to the type to search for or both.

The result of a query can be many rows. To increase performance, only the first 40 rows will be shown in the resultlist. Each time the user scrolls down to the end of the list, another 40 rows will show up and so on until the resultlist contains all rows. By selecting *Count No of Hits* it is possible to know about how many rows the resultlist finally will contain when scrolled to the end.

The resultlist has several columns and the sort order can be changed. Unwanted columns can be hidden and rows of interest can be further analysed via functions in a popupmenu. The resultlist can be expanded to cover the whole Queryform by clicking the green bar.

How is the system organized?

View: [Distribution](#) - [TopDown](#) - [System](#).

Click the + sign to expand to see what the system consists of.

How is the application organized?

View: [Distribution](#) - [TopDown](#) - [Application](#).

Click the + sign to expand to see what the application consists of.

How is the program organized?

a) If the language has programs as objects (almost every languages do):

View: [Distribution](#) - [TopDown](#) - [Object](#), filter on Objects Name.

b) If the language has programs as packages (a few languages do, for example IBM CSP/VAG45):

View: [Distribution](#) - [TopDown](#) - [Package](#), filter on Packages Name.

Click the + sign to expand to see what the program consists of.

What is the program part of?

a) If the language has programs as objects (almost all languages):

View: [Distribution](#) - [BottomUp](#) - [Object](#), filter on Objects Name.

b) If the language has programs as packages (only a few languages, for example IBM CSP/VAG45):

View: [Distribution](#) - [BottomUp](#) - [Package](#), filter on Packages Name.

Click the + sign to expand to see what the program is part of. Keep on expanding to see what next level is part of.

What is the object (class/form/screen/record/table/datafile) part of?

View: [Distribution](#) - [BottomUp](#) - [Object](#), filter on Objects Name.

Click the + sign to expand to see what the object is part of. Keep on expanding to see what next level is part of.

What is the entry (method/function/process/procedure/section) part of?

View: [Distribution](#) - [BottomUp](#) - [Entry](#), filter on Entrys Name.

Click the + sign to expand to see what the entry is part of. Keep on expanding to see what next level is part of.



From where is the object (class/form/screen/record/table/datafile) used?

View: [Dependencies](#) - [BottomUp](#) - [Object](#), filter on Objects Name

Click the + sign to expand to see where the object is used. Keep on expanding to see where next level is used. The end items of the tree structure is startingpoints.

From where is the entry (method/function/process/procedure/section) called?

View: [Dependencies](#) - [BottomUp](#) - [Entry](#), filter on Entrys Name

Click the + sign to expand to see where the entry is used. Keep on expanding to see where next level is used. The end items of the tree structure is startingpoints.

Which other objects does the object (class/form/screen/record/table/datafile) use?

View: [Dependencies](#) - [TopDown](#) - [Object](#), filter on Objects Name

Click the + sign to expand to see which other objects being used. Keep on expanding to see which objects used by next level. When all levels are expanded, all objects involved is showed.

Which other entrys does the entry (method/function/process/procedure/section) call?

View: [Dependencies](#) - [TopDown](#) - [Entry](#), filter on Entrys Name

Click the + sign to expand to see which other entrys being called. Keep on expanding to see which entrys called by next level. When all levels are expanded, all entrys involved is showed.

Show the callchain startingpoints

View: [Dependencies](#) - [BottomUp](#) - [Entry](#), filter on Entrys Name

Click the + sign to expand to see where the entry is used. Keep on expanding to see where next level is used. The end items of the tree structure is startingpoints.

Show the callchain starting at the entry

View: [Dependencies](#) - [TopDown](#) - [Entry](#), filter on Entrys Name and check Show Entrys Sequentially.

Click the + sign to expand to see which other entrys being called (in execution order). Keep on expanding to see which entrys called by next level (in execution order).. When all levels are expanded until the end, the callchain is complete.

Read code in execution order starting at the entry

View: [Dependencies](#) - [TopDown](#) - [Code](#), filter on Entrys Name.

Click the + sign to expand to see the code in the entry. A red or blue icon is showed at each call, if a + sign exists it is possible to expand and read the called code.

Middleware for Dynamic SQL

The PESO example application needs middleware settings for dynamic SQL to create the dependencies to the database tables. Add four middleware settings as below screenshots:

The screenshot shows the 'Middleware' dialog box with the following configuration:

- Choose middleware:** A list containing DynSQL_INSERT, DynSQL_SELECT, DynSQL_UPDATE, and DynSQL_XDELETE. DynSQL_INSERT is selected.
- Edit middleware settings:**
 - Name:** DynSQL_INSERT
 - Word:** INTO
 - WordType:** Word in String
 - Set ObjectIdentifier to:** (empty)
 - Offset to ObjectIdentifier Word:** 0
 - Offset to TableIdentifier Word:** 1 as Insert
 - Skip prefix on found Identifier:** (empty)
 - Add prefix on found Identifier:** (empty)
 - Skip suffix on found Identifier:** (empty)
 - Set EntryName to:** (empty)
 - Offset to EntryName Word:** 0
 - MiddleWare exists in Scope:** System named PESO
- Buttons:** + Add, - Delete, Cancel, Test, and X Close.

The screenshot shows the 'Middleware' dialog box with the following configuration:

- Choose middleware:** A list containing DynSQL_INSERT, DynSQL_SELECT, DynSQL_UPDATE, and DynSQL_XDELETE. DynSQL_SELECT is selected.
- Edit middleware settings:**
 - Name:** DynSQL_SELECT
 - Word:** FROM
 - WordType:** Word in String
 - Set ObjectIdentifier to:** (empty)
 - Offset to ObjectIdentifier Word:** 0
 - Offset to TableIdentifier Word:** 1 as Select
 - Skip prefix on found Identifier:** (empty)
 - Add prefix on found Identifier:** (empty)
 - Skip suffix on found Identifier:** (empty)
 - Set EntryName to:** (empty)
 - Offset to EntryName Word:** 0
 - MiddleWare exists in Scope:** System named PESO
- Buttons:** + Add, - Delete, Cancel, Test, and X Close.

The screenshot shows the 'MiddleWare' dialog box. On the left, under 'Choose middleware:', a list contains 'DynSQL_INSERT', 'DynSQL_SELECT', 'DynSQL_UPDATE', and 'DynSQL_XDELETE'. 'DynSQL_UPDATE' is selected. The 'Edit middleware settings:' section on the right has the following fields: 'Name:' with 'DynSQL_UPDATE' and a 'Test' button; 'Word:' with 'UPDATE' and 'WordType:' with a dropdown set to 'Word in String'; 'Set ObjectIdentifier to:' (empty); 'Offset to ObjectIdentifier Word:' with '0'; 'Offset to TableIdentifier Word:' with '1' and a dropdown set to 'Update'; 'Skip prefix on found Identifier:' (empty); 'Add prefix on found Identifier:' (empty); 'Skip suffix on found Identifier:' (empty); 'Set EntryName to:' (empty); 'Offset to EntryName Word:' with '0'; and 'MiddleWare exists in Scope:' with a dropdown set to 'System named' and a text box containing 'PESD'. On the right side, there are buttons for '+ Add', '- Delete', 'Cancel', and 'Close'.

The screenshot shows the 'MiddleWare' dialog box. On the left, under 'Choose middleware:', a list contains 'DynSQL_INSERT', 'DynSQL_SELECT', 'DynSQL_UPDATE', and 'DynSQL_XDELETE'. 'DynSQL_XDELETE' is selected. The 'Edit middleware settings:' section on the right has the following fields: 'Name:' with 'DynSQL_XDELETE' and a 'Test' button; 'Word:' with 'DELETE' and 'WordType:' with a dropdown set to 'Word in String'; 'Set ObjectIdentifier to:' (empty); 'Offset to ObjectIdentifier Word:' with '0'; 'Offset to TableIdentifier Word:' with '3' and a dropdown set to 'Delete'; 'Skip prefix on found Identifier:' (empty); 'Add prefix on found Identifier:' (empty); 'Skip suffix on found Identifier:' (empty); 'Set EntryName to:' (empty); 'Offset to EntryName Word:' with '0'; and 'MiddleWare exists in Scope:' with a dropdown set to 'System named' and a text box containing 'PESD'. On the right side, there are buttons for '+ Add', '- Delete', 'Cancel', and 'Close'.

Note that DELETE is named XDELETE to place it after SELECT in the list of middleware settings. The reason is that SELECT and DELETE statements look the same concerning the keyword FROM. The hits on SELECT **FROM** <table> made by DynSQL_SELECT will now be overwritten by the hits from **DELETE** * FROM <table> made by DynSQL_DELETE.

After setting the middlewares, choose **Tools/Load**, select option 'Rebuild all Dependencies' and click *Load* to add the new dependencies from the middleware settings.

Support

Answers to common questions and known problems can be found at

<http://www.resourceminer.nu/support.htm>
