RC
&
You

• Royal Cyber
• Development Modernization
• Newsletter

"Technical Content For A Rational World"

In this issue:

COBOL Standards Checking – RDz's "Code Review" Feature – Part I

This article is the first in a two-part series that explores the use of RDz's "Software Analyzer" for analyzing ***COBOL***** *Performance and Maintainability.*

By Chris Leland, Royal Cyber

Question…what are the four things you do to COBOL programs as you move them through the lifecycle? **1. Code. 2. Compile for syntax. 3. Test. And 4. Promote into production.** Right? Yes… Wait … Wasn't there a 5th thing? … Something about "desk-checking", or "standards reviewing", or some such step? Why sure there is… Well …. maybe there is… To be honest? We think there is – but maybe it's not painstakingly enforced… Come to think of it? What do we do around here when it comes to determining if code's been written that is; **a.** efficient, and **b.** maintainable?

If that mental conundrum hits home, then you're in the same boat with about 85% of z/OS shops out there. Who know that they ***should*** do some sort of standards-analysis or evaluation of the code that becomes an asset on the corporate balance sheet before it's thrown over the wall into production. But who don't have the time, or don't have the wherewithall to work this critical step into the lifecycle.

Fret not… help is on the way in the form of RDz's Code Review (*Software Analyzer*) product feature. But before we get into the details, let's have a look at exactly what doing this sort of thing can mean to your company.

Why do COBOL Code Reviews for Run-Time Performance?

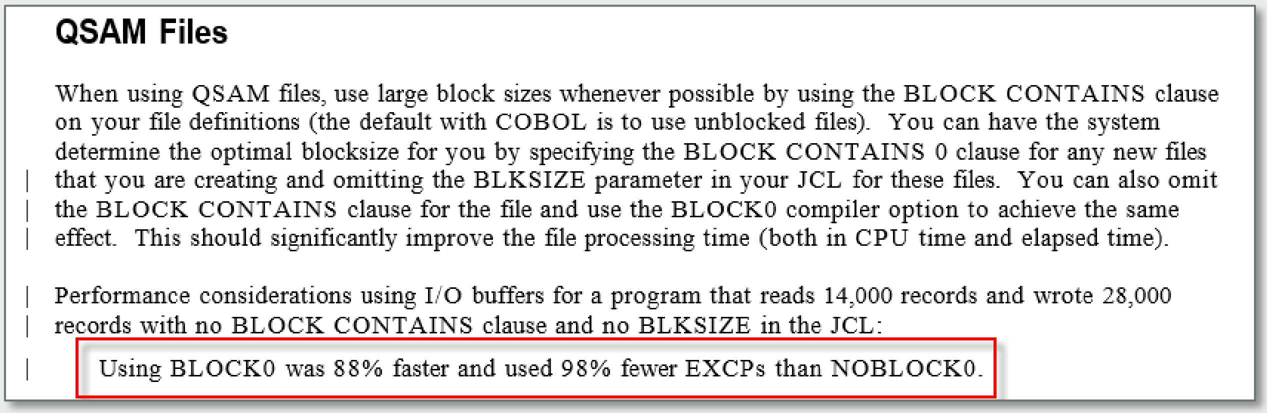
Over the years IBM has been publishing COBOL efficiency coding Redbooks, Powerpoint presentations and Tech-Notes. In fact, a few years ago they published a white paper title: ***IBM Enterprise COBOL Version 4 Release 2 Performance Tuning***, written by Rick (R.J.) Arellanes. http://www-01.ibm.com/support/docview.wss?uid=swg27018287&aid=1

Besides being one of the world's foremost experts on COBOL efficiency, Rick took the time to setup a clean room environment and benchmark potential savings from using different COBOL Compiler, LE and z/OS tuning and COBOL programming options. And then he documented the results in a well-written white paper that I strongly recommend you grab – if you're looking to save Production MIPS. "Production MIPS?" you ask… Yes.

You see – z/OS costs are typically divided into MIPS used for the **development** of your applications – and for the **production running your applications** to transact business. In almost all cases production MIPS are by far the larger percentage of data center costs. Typically something like 70/30 (production/development) or 65/35, etc. So, while it's nice to chase down development MIPS cost savings, saving Production MIPS is where the biggest bang for your buck lives. So, ensuring that you're abiding by the strategies and suggestions in Rick's white paper is key.

We've screen captured a few of the more eye-popping returns on your investment in Figures 1 through 8 below. Have a look at these before continuing on with this article. Please note that we've added the red rectangular boxes for emphasis – they're not in Rick's whitepaper. Also, at the risk of redundancy, please consider reading Rick's white paper. Because this article is not specific to COBOL efficiency, we've left out virtuall all of the detail in Rick's doc. Which shows that, when it comes to production efficiencies, there are better/worse ways to develop an application. And that the cost savings for "better" are **not** insignificant.

**** RDz's Code Review supports PL/I and Java, in addition to COBOL**



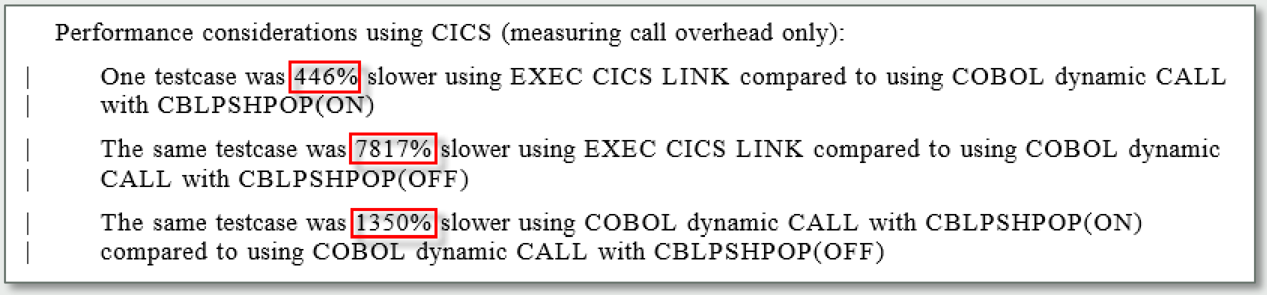
QSAM Files

When using QSAM files, use large block sizes whenever possible by using the BLOCK CONTAINS clause on your file definitions (the default with COBOL is to use unblocked files). You can have the system determine the optimal blocksize for you by specifying the BLOCK CONTAINS 0 clause for any new files that you are creating and omitting the BLKSIZE parameter in your JCL for these files. You can also omit the BLOCK CONTAINS clause for the file and use the BLOCK0 compiler option to achieve the same effect. This should significantly improve the file processing time (both in CPU time and elapsed time).

Performance considerations using I/O buffers for a program that reads 14,000 records and wrote 28,000 records with no BLOCK CONTAINS clause and no BLKSIZE in the JCL:

Using BLOCK0 was 88% faster and used 98% fewer EXCPs than NOBLOCK0.

Figure 1. Performance Benefits of Coding BLOCK CONTAINS 0



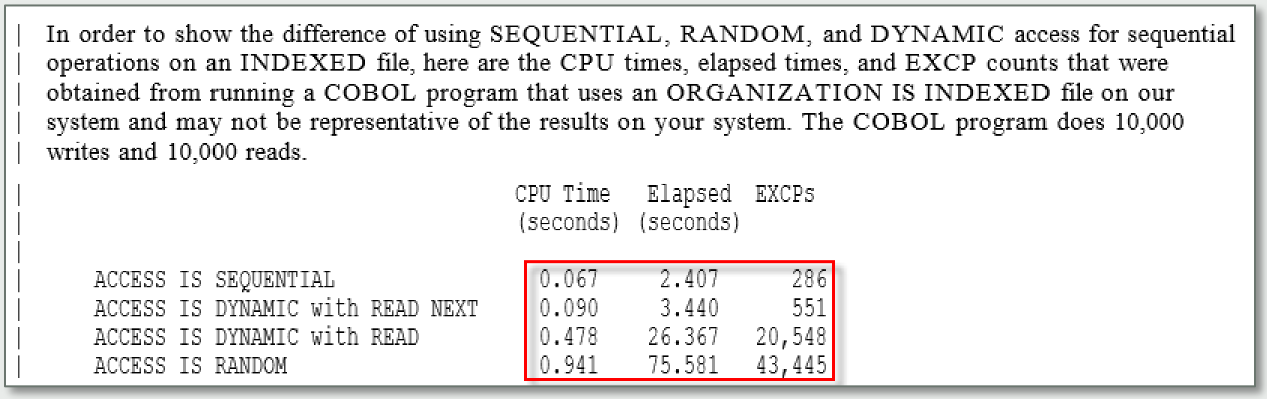
Performance considerations using CICS (measuring call overhead only):

One testcase was 446% slower using EXEC CICS LINK compared to using COBOL dynamic CALL with CBLPSHPOP(ON)

The same testcase was 7817% slower using EXEC CICS LINK compared to using COBOL dynamic CALL with CBLPSHPOP(OFF)

The same testcase was 1350% slower using COBOL dynamic CALL with CBLPSHPOP(ON) compared to using COBOL dynamic CALL with CBLPSHPOP(OFF)

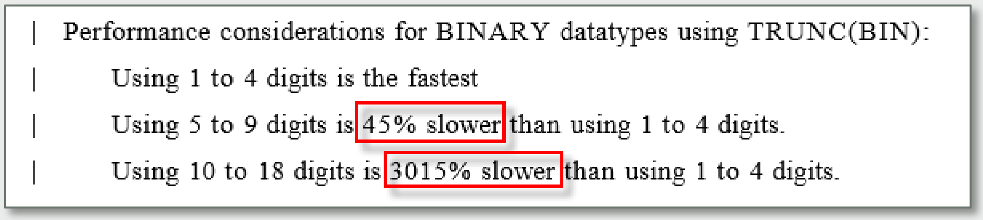
Figure 2. Performance considerations for CICS LINK vs. COBOL dynamic CALL



In order to show the difference of using SEQUENTIAL, RANDOM, and DYNAMIC access for sequential operations on an INDEXED file, here are the CPU times, elapsed times, and EXCP counts that were obtained from running a COBOL program that uses an ORGANIZATION IS INDEXED file on our system and may not be representative of the results on your system. The COBOL program does 10,000 writes and 10,000 reads.

	CPU Time (seconds)	Elapsed (seconds)	EXCPs
ACCESS IS SEQUENTIAL	0.067	2.407	286
ACCESS IS DYNAMIC with READ NEXT	0.090	3.440	551
ACCESS IS DYNAMIC with READ	0.478	26.367	20,548
ACCESS IS RANDOM	0.941	75.581	43,445

Figure 3. Performance considerations for different VSAM file access methods



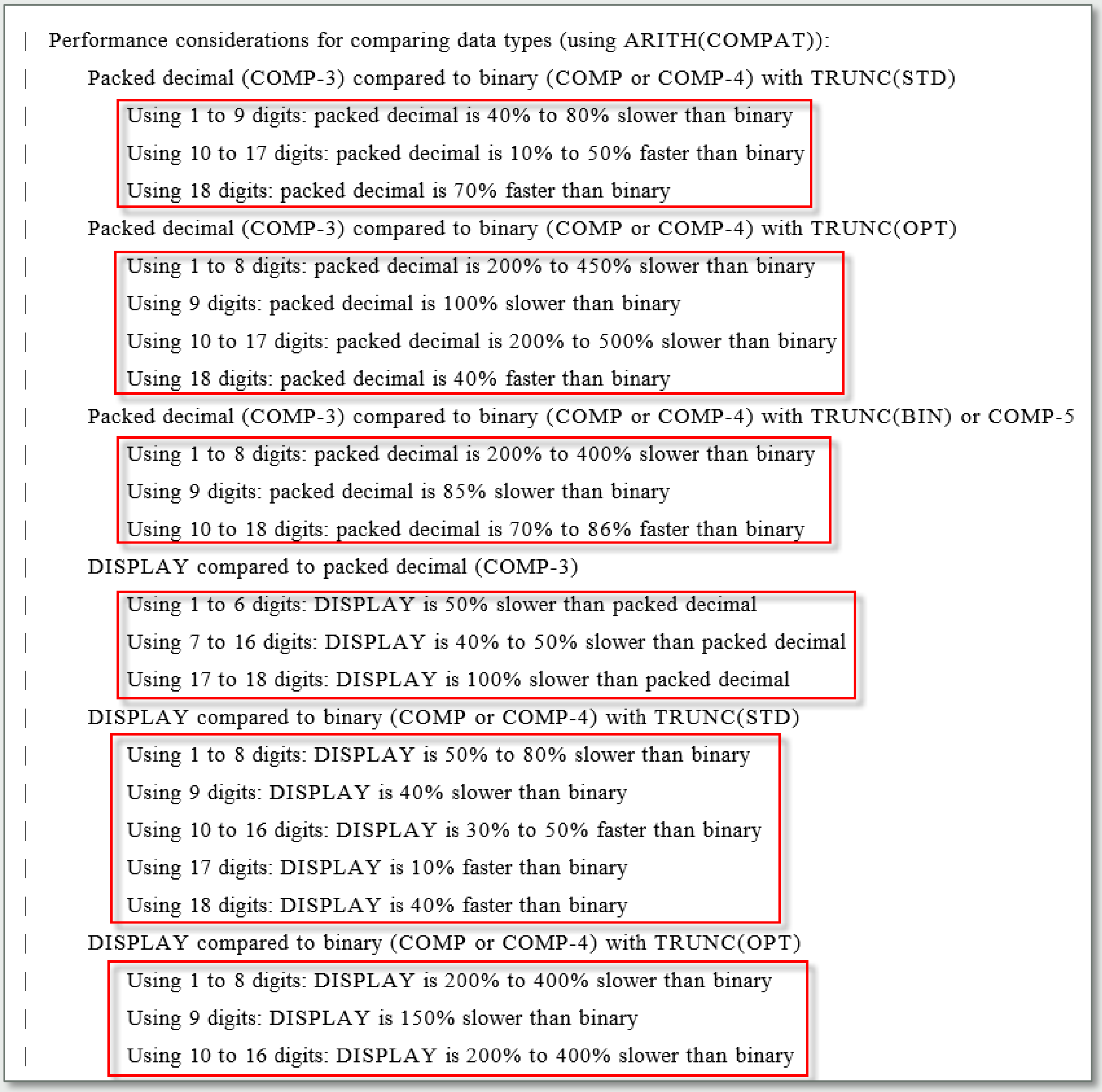
Performance considerations for BINARY datatypes using TRUNC(BIN):

Using 1 to 4 digits is the fastest

Using 5 to 9 digits is 45% slower than using 1 to 4 digits.

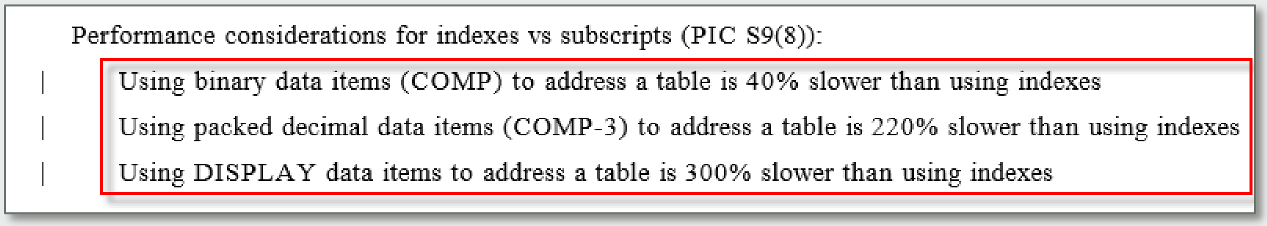
Using 10 to 18 digits is 3015% slower than using 1 to 4 digits.

Figure 4. Performance considerations for COMP (binary) data declaration



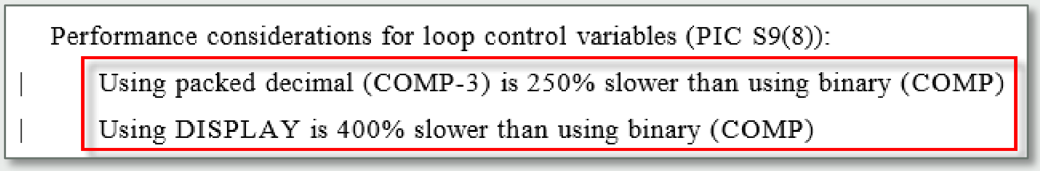
```
|  Performance considerations for comparing data types (using ARITH(COMPAT)):
|      Packed decimal (COMP-3) compared to binary (COMP or COMP-4) with TRUNC(STD)
|          Using 1 to 9 digits: packed decimal is 40% to 80% slower than binary
|          Using 10 to 17 digits: packed decimal is 10% to 50% faster than binary
|          Using 18 digits: packed decimal is 70% faster than binary
|      Packed decimal (COMP-3) compared to binary (COMP or COMP-4) with TRUNC(OPT)
|          Using 1 to 8 digits: packed decimal is 200% to 450% slower than binary
|          Using 9 digits: packed decimal is 100% slower than binary
|          Using 10 to 17 digits: packed decimal is 200% to 500% slower than binary
|          Using 18 digits: packed decimal is 40% faster than binary
|      Packed decimal (COMP-3) compared to binary (COMP or COMP-4) with TRUNC(BIN) or COMP-5
|          Using 1 to 8 digits: packed decimal is 200% to 400% slower than binary
|          Using 9 digits: packed decimal is 85% slower than binary
|          Using 10 to 18 digits: packed decimal is 70% to 86% faster than binary
|      DISPLAY compared to packed decimal (COMP-3)
|          Using 1 to 6 digits: DISPLAY is 50% slower than packed decimal
|          Using 7 to 16 digits: DISPLAY is 40% to 50% slower than packed decimal
|          Using 17 to 18 digits: DISPLAY is 100% slower than packed decimal
|      DISPLAY compared to binary (COMP or COMP-4) with TRUNC(STD)
|          Using 1 to 8 digits: DISPLAY is 50% to 80% slower than binary
|          Using 9 digits: DISPLAY is 40% slower than binary
|          Using 10 to 16 digits: DISPLAY is 30% to 50% faster than binary
|          Using 17 digits: DISPLAY is 10% faster than binary
|          Using 18 digits: DISPLAY is 40% faster than binary
|      DISPLAY compared to binary (COMP or COMP-4) with TRUNC(OPT)
|          Using 1 to 8 digits: DISPLAY is 200% to 400% slower than binary
|          Using 9 digits: DISPLAY is 150% slower than binary
|          Using 10 to 16 digits: DISPLAY is 200% to 400% slower than binary
```

Figure 5. Performance considerations for DISPLAY and COMP-3 vs. COMP declarations



```
     Performance considerations for indexes vs subscripts (PIC S9(8)):
|        Using binary data items (COMP) to address a table is 40% slower than using indexes
|        Using packed decimal data items (COMP-3) to address a table is 220% slower than using indexes
|        Using DISPLAY data items to address a table is 300% slower than using indexes
```

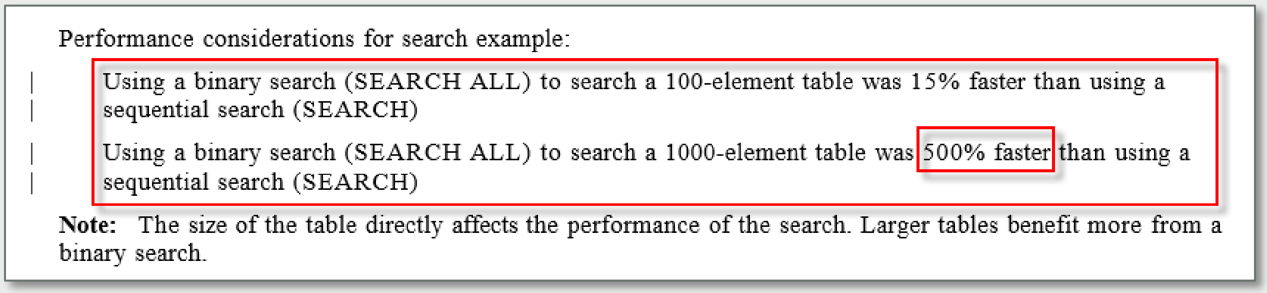
Figure 6. Performance considerations for table subscripting


Performance considerations for loop control variables (PIC S9(8)):

| Using packed decimal (COMP-3) is 250% slower than using binary (COMP)

| Using DISPLAY is 400% slower than using binary (COMP)

Figure 7. Performance considerations for iterative structures (PEFORM VARYING…)


Performance considerations for search example:

| Using a binary search (SEARCH ALL) to search a 100-element table was 15% faster than using a
| sequential search (SEARCH)

| Using a binary search (SEARCH ALL) to search a 1000-element table was 500% faster than using a
| sequential search (SEARCH)

Note: The size of the table directly affects the performance of the search. Larger tables benefit more from a binary search.

Figure 8. Performance considerations for sequential vs. binary search (SEARCH vs. SEARCH ALL)

Code Review Tools

So, I'm betting that Rick's numbers have gotten your attention (didn't expect **quite** that level of disparity between coding options, eh?) If you're convinced now of the potential savings for coding COBOL with an eye towards efficiency, the next question to be answered is, "how" – how are you going to find violations of coding constructs that – when recast more efficiently can save you production MIPS …lots of production MIPS.

But that's not where the savings end. Code Review for program "maintain-ability" through readable operational code, quality inline documentation, well-named procedures/paragraphs and variables can save programming costs, maintenance costs, and production support costs. There's a lot of upside to Code Review, keep reading.

Manual Code Review

Back in the day "Desk-Checking" methods were used – where project teams examined programs manually. But over the years Desk-Checking has fallen out of favor, due to the high capital costs of teams reading code and the requirement to utilize ("tie up") your very best / most experienced COBOL experts in this demanding process. Enter Code Review tools.

Code Review tools parse and inspect program source – and return some type of reporting on the quality of the statements. Benefits of using Code Review tooling include:

- **Lowered developer resource costs**
- **Accuracy** - more issues found - it's easy to miss things in source – as we all can attest
- **Can schedule scans to be run in batch** – for certain tools
- **And can enforce issues to be** resolved – when scans are run in batch (this is tool-dependent)
- **Can configure scanning options to include/exclude certain rules**
- **Can scan source code Deltas-only** - in other words, you "baseline" the code, and run the scan only against new versions of the program

Besides Performance Code Review, most tools offer code inspection on the level of "**maintainability**" or structured coding standards; no ALTER statements, GO TO statements only pointing to paragraph exits, etc.

And finally, when using RDz – Code Review functionality provides **convenience** – as you can finish doing your work and fire off a scan at any time from a context menu.

The downside to using automated tooling for Code Review includes – but is not limited to the following:

- **An abundance of "False Positives."** Where the tool catches what it considers a code violation when in fact – within the application/business context - what is coded could be the best (or only) solution
- **Missed code violations**. The rule violation reports produced are only as good as the software doing the parsing. And COBOL is a complex language. Little things such as splitting statements over multiple lines can sometimes fool products.
- **Attenuation of expertise**. When you turn COBOL efficiency coding standards over to a software package, programmers become dependent on that package – and might never bother to pick up efficiency-coding-insight and skills. After some period of time, who's minding the store? Oh yeah… Rick Arellanes ☺
- **Loss of accuracy over time.** Or your tools do not keep up with the latest COBOL Compiler "best practices". Back in the mid-1980's the DB2 Optimizer was growing up fast. And with each release some percentage of SQL efficiency coding rules-of-thumb became obsolete. So SQL coding practices that might have saved money at DB2 version 1.7 actually cost more money @ DB2 version 2.1. Etc.

But in spite of the above, most shops are looking to utilize automated Code Review tool to manage the process of standards checking – for all the benefits cited earlier. Enter RDz's Code Review feature: **Software Analyzer.**

RDz's Code Review/Software Analyzer

At release 8.0.3 (circa 2011) IBM GA'd a function off of the RDz Context Menu named: "***Software Analyzer***" – essentially a euphemism for ***"this is RDz's code review functionality"***. The feature was lifted from long-standing Java/eclipse static code analysis available in IBM/RAD since the 1950's. Okay – not quite that long – but it's been around long enough for there to be ~211 Java Code Review rules in the box. By the way this kind of code-reuse larceny is fairly common in Java development software @IBM – and why not? As long as it benefits you.

The first Software Analyzer release was COBOL-only (COBOL plus Java). And it was a little thin on both functionality/flexibility (not many use cases) and Code Review depth (not many rules in the initial release). In v8.0.3 you could run Code Review on individual COBOL programs open from either Local Workstation projects or from z/OS through Remote Systems or in the CARMA view. And you could also drag an entire PDS to a Local Workstation and run Code Review against all of the COBOL programs in the PDS as one/single action.

Right out of the gate several things were apparent about the initial release's functionality:

- **Because IBM borrowed from mature Java technology, the design and implementation worked (well)**
- **The U.I. was extremely easy to use/interpret results/interact with**
- **The implementation of Code Review process was simple**
- **But there was no way to implement custom Code Review rules in the initial release**
- **Everything was executed (interactively) on the client**
    - **Good - in that this saved mainframe MIPS and made the function easy to use**
    - **Bad - in that the Code Review process is best centralized – not federated out**
- **And support for running Code Review against program source Deltas was not available**

Over the course of the next few years IBM kept improving Code Review. So that today (@v9.0.1.2) the following is the state of things:

- **Code Review is still easy for developers to use**
- **There are now ~55 out of the box rules (see Table 1)**
- **You now can code your own custom COBOL Code Review rules**
- **PL/I is supported**
- **You can now execute Code Review interactively (on the client) or through batch JCL – on z/OS**

So all-in-all, what began as positive add-on to RDz for a very good cause (recall Figures 1 – 8) is now extremely robust and in our view, **world class** – with only one or two missing options which we'll get to later in this article.

RDz's Code Review Concepts/Taxonomy

RDz's Code Review functionality consists of **Analysis Configuration**, **individual Code Review Rules** and **Rulesets**. Rulesets are simply named organizers for a collection of related individual Code Review rules. They're basically like Windows folders used to organize and manage individual files on your hard-drive. For example, a Ruleset named: ***Performance Rules*** – could have several Code Review rules to scan your programs for run-time efficiency problems such as those in Figures 1 ➔ 8. ***COBOL v5 Compatibility Rules*** could be run to test for COBOL statements that are either deprecated in COBOL version 5 or will not compile. You can create any number of custom Rulesets if it helps you to understand and use them effectively in your SDLC.

An Analysis Configuration is a named collection of Rulesets that you select to run against your code. You can have one or you can have many Analysis Configurations. As with Rulesets, you can name your Analysis Configurations whatever you like in order to make their use understandable and straightforward. Some examples we've seen for organizing Analysis Configurations are; By SDLC cycle: (*Development*, *Q.A.*, *Prod*, etc.), or – by business division (*Group Claims, Medicare, Property Casualty*, etc.). The relationship among Analysis Configurations, Rulesets and individual Code Review rules is shown in **Figure 9**. This taxonomy provides an extremely flexible means of organizing your Code Review work.

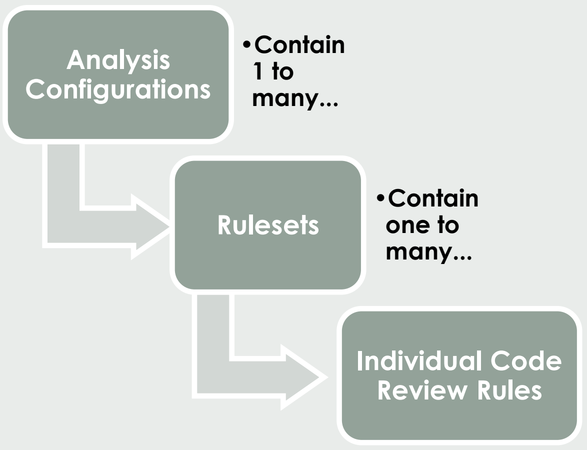
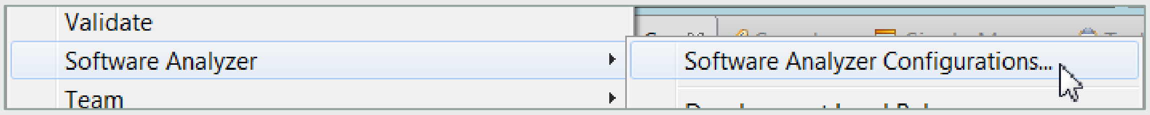


Figure 9. Relationship among RDz code Review artifacts

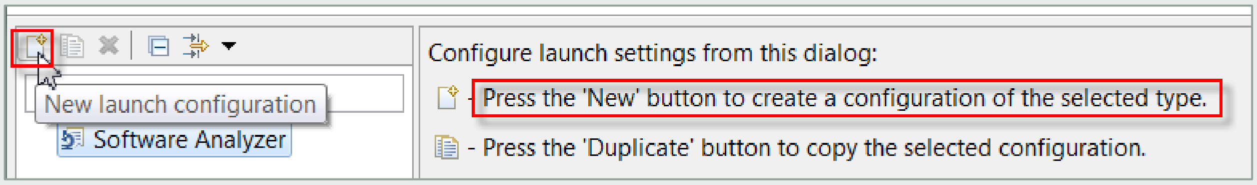
Creating Your First Code Review Rules

You begin by defining an Analysis Configuration. This is done from the **Context Menu** within edit on a program - or from the **Run** menu on the toolbar, under **Analysis**. Let's assume you're in the Editor:

From edit, select: **Software Analyzer >** **Software Analyzer Configurations…**



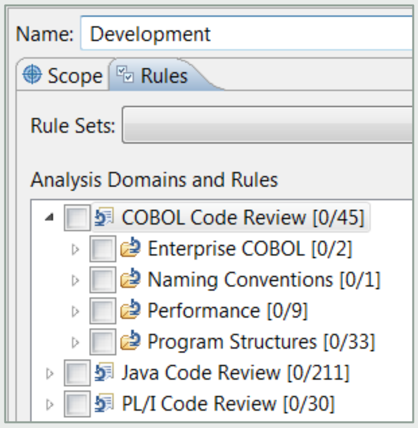
You will be presented with the New launch configuration wizard. From this wizard you select **Software Analyzer** and click the: **New launch configuration** button.



Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.

- Press the 'Duplicate' button to copy the selected configuration.


Name: Development

Scope | Rules

Rule Sets:

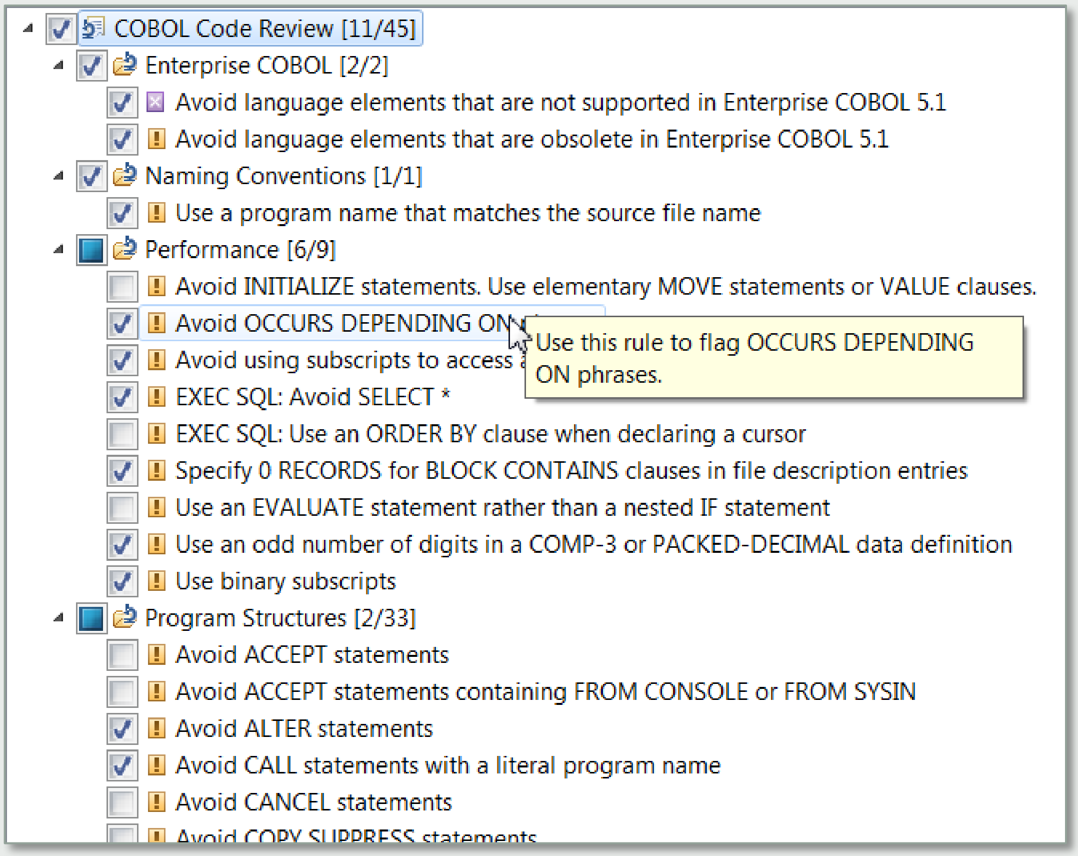
Analysis Domains and Rules

- ☐ COBOL Code Review [0/45]
  - ☐ Enterprise COBOL [0/2]
  - ☐ Naming Conventions [0/1]
  - ☐ Performance [0/9]
  - ☐ Program Structures [0/33]
- ☐ Java Code Review [0/211]
- ☐ PL/I Code Review [0/30]

This opens an area for you to name your Analysis Configuration (I've named mine Development) and select rules from the COBOL Code Review **Analysis Domains and Rules** – which are divided into four Rulesets (categories):

1. **Enterprise COBOL –** these two rules catch any statements in your code that are incompatible with Enterprise COBOL v5
2. **Naming Conventions –** there's a single rule in this Ruleset that will catch any COBOL file where the PROGRAM-ID does not match the PDS member name
3. **Performance –** there are 9 rules under the Performance Ruleset
4. **Program Structures –** and there are 33 rules in the Program Structures Ruleset… which evaluate the maintainability of your COBOL code

To select a rule for inclusion in my "Development" Analysis Configuration I've expanded the Ruleset twisties (little downward triangles), and checked the box next to the rule. If you're unsure of the meaning of a given Rule mouse-over the Rule (see **Figure 10** below). Speaking of (what these rules mean)? When you fire off a Software Analysis RDz invokes the selected Rules/Rulesets. Any statement in your COBOL program that violates a rule will get flagged when you run the Analysis Configure (see **Figure 11**).


- ✓ COBOL Code Review [11/45]
  - ✓ Enterprise COBOL [2/2]
    - ✓ Avoid language elements that are not supported in Enterprise COBOL 5.1
    - ✓ Avoid language elements that are obsolete in Enterprise COBOL 5.1
  - ✓ Naming Conventions [1/1]
    - ✓ Use a program name that matches the source file name
  - ■ Performance [6/9]
    - ☐ Avoid INITIALIZE statements. Use elementary MOVE statements or VALUE clauses.
    - ✓ Avoid OCCURS DEPENDING ON
    - ✓ Avoid using subscripts to access
    - ✓ EXEC SQL: Avoid SELECT *
    - ☐ EXEC SQL: Use an ORDER BY clause when declaring a cursor
    - ✓ Specify 0 RECORDS for BLOCK CONTAINS clauses in file description entries
    - ☐ Use an EVALUATE statement rather than a nested IF statement
    - ✓ Use an odd number of digits in a COMP-3 or PACKED-DECIMAL data definition
    - ✓ Use binary subscripts
  - ■ Program Structures [2/33]
    - ☐ Avoid ACCEPT statements
    - ☐ Avoid ACCEPT statements containing FROM CONSOLE or FROM SYSIN
    - ✓ Avoid ALTER statements
    - ✓ Avoid CALL statements with a literal program name
    - ☐ Avoid CANCEL statements
    - ☐ Avoid COPY SUPPRESS statements

Use this rule to flag OCCURS DEPENDING ON phrases.

Figure 10. Out-of-the-Box COBOL Code Review Rules

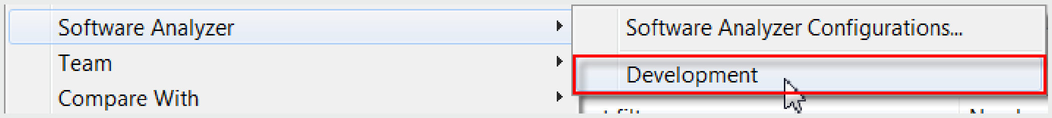
After you've selected the rules you want to run against your code:

Click **Close**, and at "**Save changes**?" click: **Yes**.

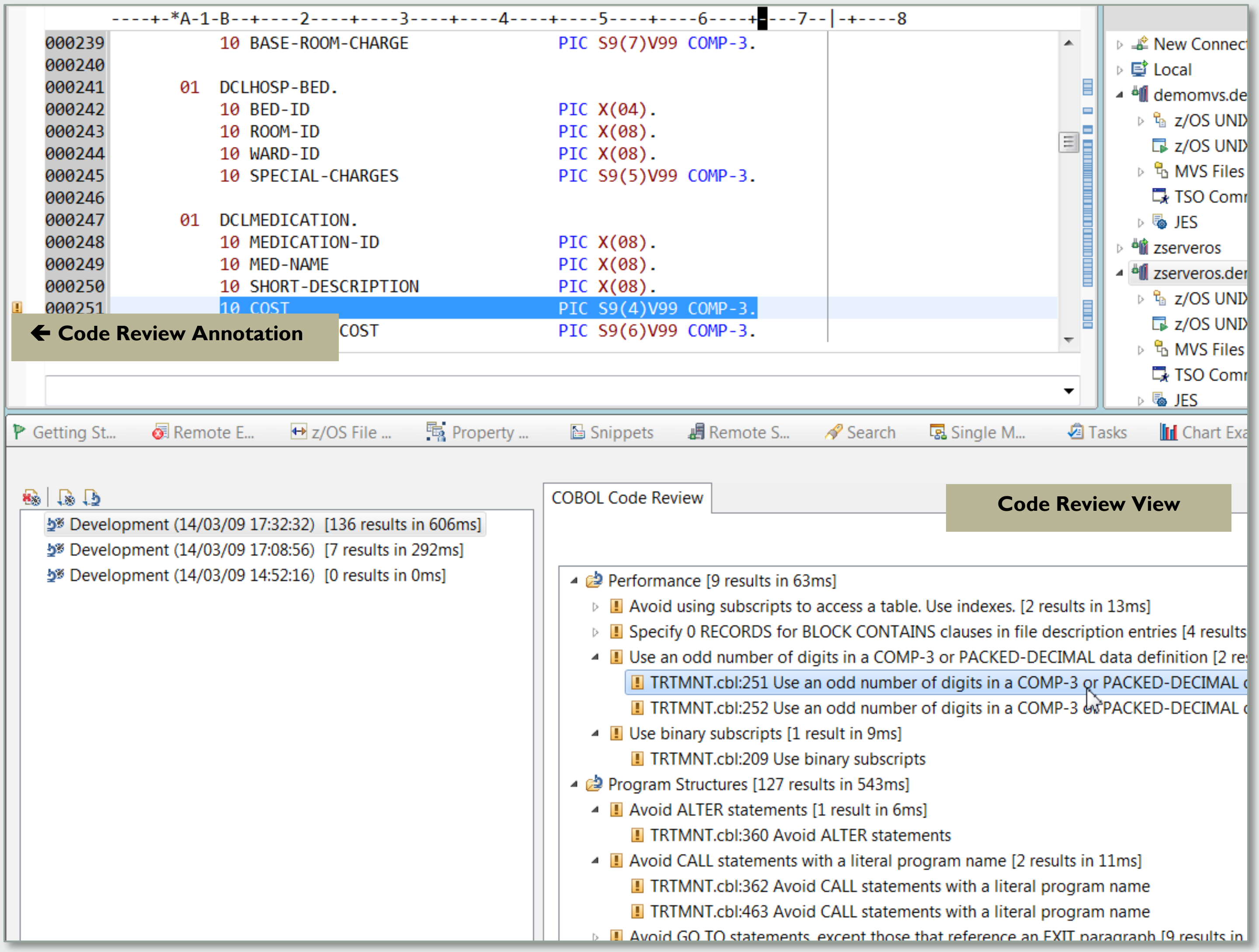
Running the Code Review Rules

When you've saved your changes and are ready to run (again - assuming that you're in Edit) select:

Software Analyzer > Development *(or the name of your Configuration)*



This will invoke Code Review on your program. Code Review analyzes your program and flags any statement that violates one or more of the rules you've selected with a Code Review annotation (see **Figure 11**).



```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
000239           10 BASE-ROOM-CHARGE               PIC S9(7)V99 COMP-3.
000240
000241       01  DCLHOSP-BED.
000242           10 BED-ID                         PIC X(04).
000243           10 ROOM-ID                        PIC X(08).
000244           10 WARD-ID                        PIC X(08).
000245           10 SPECIAL-CHARGES                PIC S9(5)V99 COMP-3.
000246
000247       01  DCLMEDICATION.
000248           10 MEDICATION-ID                  PIC X(08).
000249           10 MED-NAME                       PIC X(08).
000250           10 SHORT-DESCRIPTION              PIC X(08).
000251           10 COST                           PIC S9(4)V99 COMP-3.
                           COST                    PIC S9(6)V99 COMP-3.
```

Development (14/03/09 17:32:32) [136 results in 606ms]
Development (14/03/09 17:08:56) [7 results in 292ms]
Development (14/03/09 14:52:16) [0 results in 0ms]

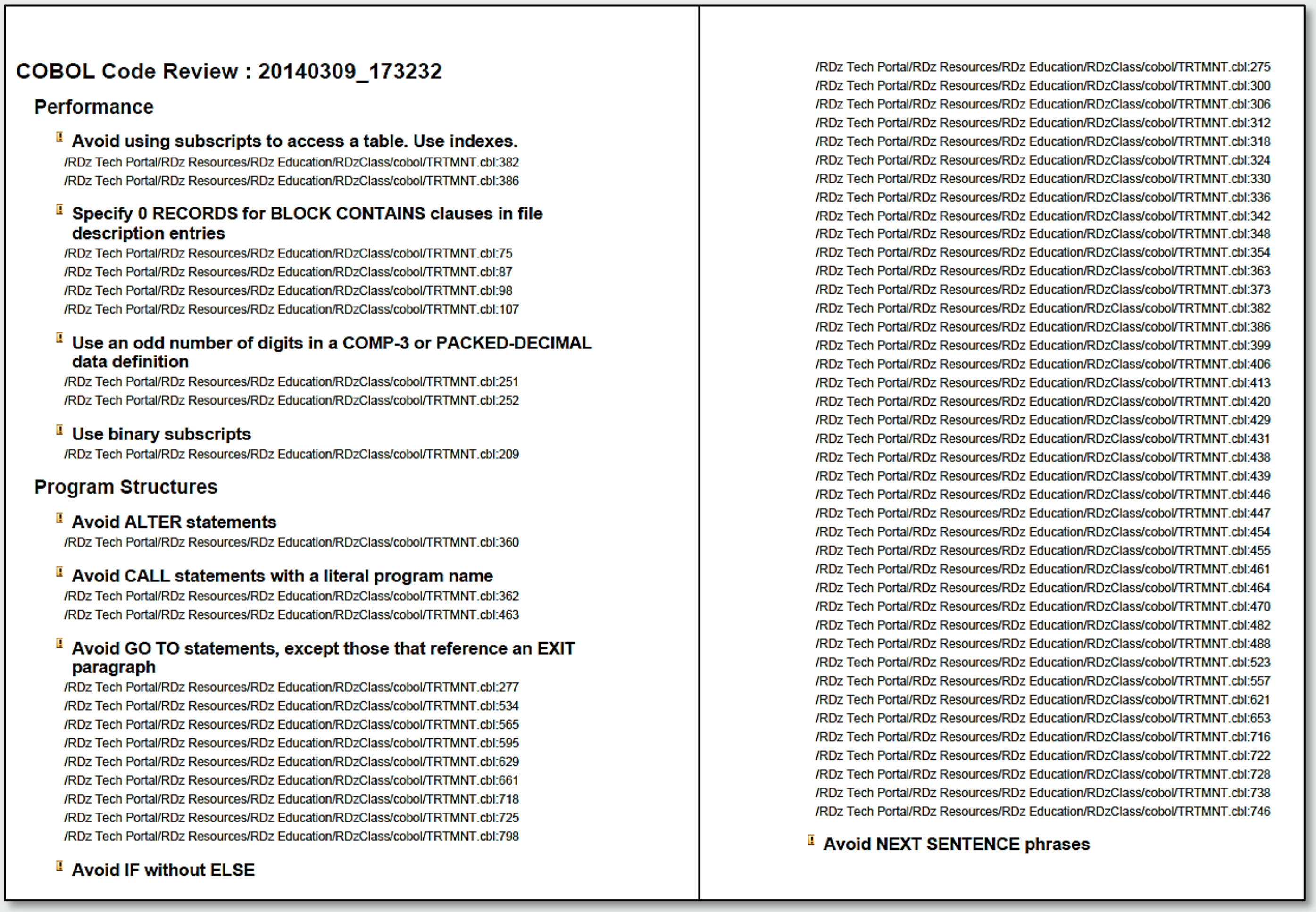
- Performance [9 results in 63ms]
  - Avoid using subscripts to access a table. Use indexes. [2 results in 13ms]
  - Specify 0 RECORDS for BLOCK CONTAINS clauses in file description entries [4 results
  - Use an odd number of digits in a COMP-3 or PACKED-DECIMAL data definition [2 re
    - TRTMNT.cbl:251 Use an odd number of digits in a COMP-3 or PACKED-DECIMAL
    - TRTMNT.cbl:252 Use an odd number of digits in a COMP-3 or PACKED-DECIMAL
  - Use binary subscripts [1 result in 9ms]
    - TRTMNT.cbl:209 Use binary subscripts
- Program Structures [127 results in 543ms]
  - Avoid ALTER statements [1 result in 6ms]
    - TRTMNT.cbl:360 Avoid ALTER statements
  - Avoid CALL statements with a literal program name [2 results in 11ms]
    - TRTMNT.cbl:362 Avoid CALL statements with a literal program name
    - TRTMNT.cbl:463 Avoid CALL statements with a literal program name
  - Avoid GO TO statements except those that reference an EXIT paragraph [9 results in

Figure 11. Rule Violation annotations – and hyperlinks from a Code Review view

After it completes Code Review, RDz aggregates all of the rule violations for you, and presents them in a Code Review view (**Figure 11**).

The code Review View organizes rule violations in your program by: Ruleset ➔ Rule ➔ Individual statement rule violations – which are hyperlinked back into the source file for quick access/analysis and statement modification (i.e. no need to do find/change commands) – just click the link to the statement in the Code Review view.

In this workflow model, you would fix (or choose not to fix) specific violations in the program, save the code, and then re-run the Code Review Analysis Configuration – iterating over the process until you're satisfied with the state of your code (or until you run out of time in your project). Note that you can also save the Code Review itself to hardcopy – either HTML or (as shown in **Figure 12**) a PDF.



COBOL Code Review : 20140309_173232

Performance

Avoid using subscripts to access a table. Use indexes.
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:382
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:386

Specify 0 RECORDS for BLOCK CONTAINS clauses in file description entries
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:75
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:87
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:98
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:107

Use an odd number of digits in a COMP-3 or PACKED-DECIMAL data definition
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:251
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:252

Use binary subscripts
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:209

Program Structures

Avoid ALTER statements
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:360

Avoid CALL statements with a literal program name
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:362
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:463

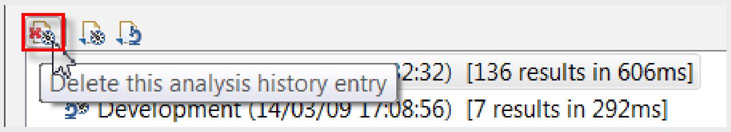
Avoid GO TO statements, except those that reference an EXIT paragraph
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:277
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:534
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:565
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:595
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:629
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:661
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:718
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:725
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:798

Avoid IF without ELSE
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:275
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:300
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:306
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:312
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:318
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:324
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:330
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:336
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:342
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:348
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:354
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:363
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:373
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:382
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:386
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:399
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:406
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:413
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:420
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:429
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:431
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:438
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:439
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:446
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:447
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:454
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:455
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:461
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:464
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:470
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:482
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:488
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:523
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:557
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:621
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:653
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:716
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:722
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:728
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:738
/RDz Tech Portal/RDz Resources/RDz Education/RDzClass/cobol/TRTMNT.cbl:746

Avoid NEXT SENTENCE phrases

Figure 12. Rule Violation annotations – saved to a PDF

If at any point in the Code Review workflow process you have no use for a particular Analysis you can delete by selecting it in the Code Review View and clicking a red "delete" button. This deletes the Analysis and removes the Annotations from the program.



Delete this analysis history entry
2:32) [136 results in 606ms]
Development (14/03/09 17:08:56) [7 results in 292ms]

About These out-of-the-box Code Review Rules … aren't some of them contradictory?

Take a close look at the ~55 out-of-the-box COBOL Code Review Rules (see Table 1). A cursory inspection shows that there are the following:

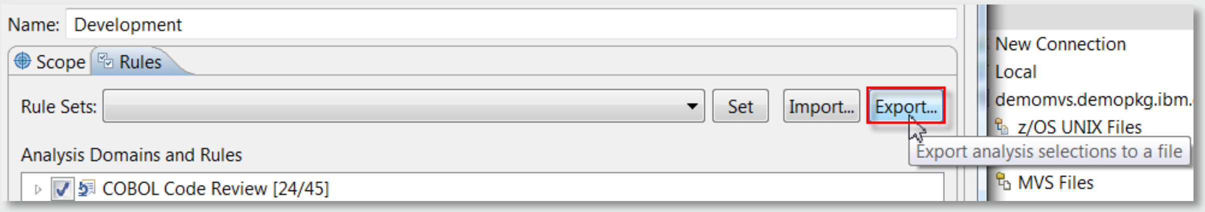
- Some very good rules
- Some not-so-good rules
- Some contradictory rules

How did this happen?

We were curious ourselves, but in listening to the RDz Product Managers discuss their approach to delivering the Rulesets they said that, in essence – companies wrote in requirements for certain rules and the RDz team created the rules – basically as stated… whether or not the rules would be acceptable to the Rick Arellaneses of the world. So it becomes incumbent on you to sift through the rules, find the ones that you agree with, check the boxes for the rules, save, test, test, and test some more, and then rollout your Code Review implementation. How does one do that (you ask)? Good question.

Rolling out and Administering Code Review Rules

The Software Configuration wizard has an Export… button on it, to save off a an Analysis Configuration to a Windows file server, and an Import… button that developers can use to fetch Analysis Configurations into their Workspace.



You can also build Code Review rules into a custom or template workspace. And finally, RDz's Push-to-Client functionality – a topic we'll cover in a future Royal Cyber Newsletter.

Our Take…

Unless you're already taking advantage of a quality automated Code Review product, take a hard look at RDz's Software Analyzer. As you've seen in this Part I article, the functionality is solid – the integration is first-rate and the out-of-the-box rules are an excellent start.

You will absolutely want to supplement these out-of-the-box rules with implementations of your shop's own custom COBOL Code Review rules, and in Part II of this article we will discuss how to do that, and we'll also cover running Code Review in batch and the parameter-driven Code Review rules available in **Preferences > Software Analyzer**. **It is worth pointing out that creating custom COBOL Code Review rules is non-trivial. It requires a combination of Eclipse/Java and advanced COBOL knowledge. This is something that Royal Cyber specializes in. We offer a formal program for COBOL Custom Code Review Rule creation, where we come on-site, show you how custom rules are implemented, and create a number of them for you during the process. Here's a link to our catalog with a detailed description to this offering**. Royal Cyber RDz Services – Q2 2014

Next steps…

RDz is one of the primary strengths of Royal Cyber. In 2014 IBM chose Royal Cyber to manage its RDz "Distance Learning" program - out of all the RDz business partners world-wide. We offer deep and quality service engagements in every phase or RDz – from Installation and Deployment, to Training/Mentoring and adoption, to administering and supporting both the RDz client and server – and finally, to evaluating your RDz Adoption and Return-on-Investment.

- For details on the Royal Cyber RDz Rollout and project/task usage modeling contact us: inquires@royalcyber.com
- To sign up for Royal Cyber RDz Free Distance Learning: http://royalcyber.com/royal-cyber-rdz-distance-learning-training-schedule/
- To get a copy of the January 2014 Newsletter – including additional deep-dive articles on RDz and RAAi: http://royalcyber.org/mn/index.html

Chris Leland: Chris is Royal Cyber's primary RDz technical instructor - having worked full-time with RDz for almost four years; developing applications in COBOL, teaching RDz classes and mentoring in COBOL and Assembler, installing and configuring RDz, delivering custom workspace design sessions, writing an RDz evaluation exam, and integrating RDz with Rational Asset Analyzer. Chris passed the IBM/RDz Certification exam with flying colors in 2011.

Code Review Rule	Description…
Avoid ACCEPT statements	Use this rule to flag all ACCEPT statements.
Avoid ACCEPT statements containing FROM CONSOLE or FROM SYSIN	Use this rule to flag ACCEPT statements that contain the phrase FROM CONSOLE or FROM SYSIN.
Avoid ALTER statements	Use this rule to flag ALTER statements.
Avoid CALL statements with a literal program name	Use this rule to flag CALL statements that specify the program name as a literal.
Avoid calling the specified routine	Use this template to flag CALL statements that call a specified routine. Enter the name of the routine as a parameter to this template.
Avoid CANCEL statements	Use this rule to flag CANCEL statements.
Avoid COPY SUPPRESS statements	Use this rule to flag COPY statements that contain the SUPPRESS phrase.
Avoid CORRESPONDING phrases	Use this rule to flag: ADD, SUBTRACT, and MOVE statements that contain a CORRESPONDING phrase.
Avoid DISPLAY statements containing UPON CONSOLE	Use this rule to flag DISPLAY statements that contain UPON CONSOLE.
Avoid ENTRY statements	Use this rule to flag ENTRY statements.
Avoid EXIT PROGRAM statements	Use this rule to flag EXIT PROGRAM statements.
Avoid GO TO statements	Use this rule to flag all GO TO statements.
Avoid GO TO statements, except those that reference an exit paragraph	Use this rule to flag all GO TO statements, except those that transfer control to an exit paragraph. An exit paragraph is a paragraph containing only an EXIT statement.
Avoid IF without ELSE	Use this rule to flag IF statements that do not contain an ELSE clause.
Avoid including the specified copy book	Use this template to flag COPY statements that refer to a specified copy book. Enter the name of the copy book as a parameter to this template.
Avoid INITIALIZE statements. Use elementary MOVE statements or VALUE clauses.	Use this rule to flag INITIALIZE statements.
Avoid nesting IF statements deeper than the specified number of levels	Use this template to flag IF statements that are nested deeper than a specified number of levels. Enter the number of levels of nesting as a parameter to this template.
Avoid NEXT SENTENCE phrases	Use this rule to flag all NEXT SENTENCE phrases.
Avoid OCCURS DEPENDING ON phrases	Use this rule to flag OCCURS DEPENDING ON phrases.
Avoid PERFORM, except PERFORM section	Use this rule to flag all PERFORM statements, except those that contain sections only.
Avoid RESERVE clauses in FILE-CONTROL paragraphs	Use this rule to flag RESERVE clauses in FILE-CONTROL paragraphs.
Avoid static calls except for the specified program name	Use this template to flag CALL statements in which (a) a Java static method or a COBOL factory method is called, and (b) the method name does not match the specified program name. Enter the program name as a parameter to the template.
Avoid STOP RUN and STOP literal statements	Use this rule to flag STOP RUN and STOP literal statements.
Avoid THRU phrases in PERFORM statements	Use this rule to flag PERFORM statements that include a THRU phrase.
Avoid using level-88 entries in data descriptions	Use this rule to flag data descriptions that use level-88 entries. Each level-88 entry is flagged.
Avoid using more than one EXIT statement per section	Use this rule to flag sections that contain more than one EXIT statement.
Avoid using SECTION in a procedure division	Use this rule to flag any SECTION declarations in the procedure division.

Avoid using subscripts to access a table. Use indexes.	Use this rule to flag any data item (a) that is used as a subscript to access a table element, and (b) that is not specified in an INDEXED BY phrase in the OCCURS clause that defines the table.
Avoid using the selected compiler directives	Use this template to flag the selected compiler directives. Select the compiler directives to flag: BASIS, CBL (PROCESS), COPY, EJECT, REPLACE, SERVICE LABEL, SERVICE RELOAD, SKIP, and TITLE.
Avoid XML-PARSE statements	Use this rule to flag XML-PARSE statements.
EXEC CICS: Check EIBRESP after NOHANDLE	Use this rule to flag any EXEC CICS statement that specifies the NOHANDLE option and is not followed by an IF statement or an EVALUATE statement that checks the value of EIBRESP.
EXEC CICS: Use DFHRESP to check the return value	This rule applies to data items that are used as the RESP or RESP2 parameter of an EXEC CICS command.
EXEC CICS: Use the RESP option	Use this rule to flag CICS EXEC commands that do not include the RESP option.
EXEC SQL: Check SQLCODE	Use this rule to flag any EXEC SQL statement that is not followed by an IF statement or an EVALUATE statement that checks the value of SQLCODE.
EXEC SQL: Use a WHERE clause in selected statements	Use this template to flag any EXEC SQL statement containing a SELECT, DELETE, or UPDATE statement that does not include a WHERE clause. Select the types of statement to flag: SELECT, DELETE, and UPDATE.
EXEC SQL: Avoid SELECT *	Use this rule to flag EXEC SQL statements that contain a SELECT * statement.
EXEC SQL: Use an ORDER BY clause when declaring a cursor	Use this rule to flag EXEC SQL statements that declare a cursor without specifying an ORDER BY clause in the contained SELECT statement.
Follow the specified naming convention for COBOL file names	Use this template to flag program file names that do not match a specified regular expression. Enter the regular expression as a parameter to this template. The comparison ignores the file extension of the program name, if any.
Inline PERFORM statement cannot exceed the specified line number limit	Use this template to flag PERFORM statements that contain more than a specified number of lines. Enter the number of lines as a parameter to this template.
Procedure division statements cannot exceed the specified line number limit	Use this template to flag PROCEDURE divisions that contain more than a specified number of lines. Enter the number of lines as a parameter to this template.
Specify 0 RECORDS for BLOCK CONTAINS clauses in file description entries	Use this rule to flag BLOCK CONTAINS clauses that do not specify 0 RECORDS.
Use a program name that matches the source file name	Use this rule to flag any PROGRAM-ID division whose program name is different than its source file name. The file extension, if any, of the source file is excluded from the comparison.
Use a scope terminator phrase with the specified COBOL statement type	Use this template to flag the specified type of COBOL statement if it does not contain a scope terminator phrase. Select the type of COBOL statement as a parameter for the template.
Use a WHEN OTHER phrase with an EVALUATE statement	Use this rule to flag EVALUATE statements that do not include a WHEN OTHER phrase.
Use an EVALUATE statement rather than a nested IF statement	Use this rule to flag nested IF statements. Note that if the nesting is more than one level deep then the rule flags only the outermost nested IF statement.
Use an exit paragraph in each section	Use this rule to flag sections that do not contain an exit paragraph. An exit paragraph is a paragraph containing only an EXIT statement.
Use an odd number of digits in a COMP-3 or PACKED-DECIMAL data definition	Use this rule to flag any data definition that is declared as type COMP-3 or PACKED-DECIMAL and does not contain an odd number of digits.

Use binary subscripts	Use this rule to flag any data item (a) that is used as a subscript to access a table element, and (b) that is not declared with a usage of COMP, COMPUTATIONAL, or BINARY.
Use comments to describe all paragraphs	Use this template to flag paragraph declarations that are not immediately preceded or followed by a comment. Select whether the comment precedes or follows the paragraph declaration.
Use comments to describe all sections	Use this template to flag section headers that are not immediately preceded or followed by a comment. Select whether the comment precedes or follows the section header.
Use CONTINUE rather than NEXT SENTENCE inside a scoped range	Use this rule to flag NEXT SENTENCE statements that lie within the scope of any statement that has an explicit scope terminator. For example, a NEXT SENTENCE statement is flagged if it lies between an IF statement and its corresponding END-IF phrase.
Use CURRENT-DATE rather than ACCEPT DATE or ACCEPT TIME	Use this rule to flag ACCEPT DATE and ACCEPT TIME statements.
Use level numbers in the sequence 01, 05, 10, 15, ...	Use this rule to flag data structure definitions containing level numbers (a) that are not in ascending sequence; or (b) than do not have a value of either 1 or a multiple of 5.
Use SEARCH ALL rather than SEARCH to search a table	Use this rule to flag table searches that use SEARCH rather than SEARCH ALL.
Use the specified prefix with condition names	Use this template to flag data descriptions that do not begin with the specified prefix. Enter the prefix as a parameter to the template.
Use THRU phrases in PERFORM statements	Use this rule to flag PERFORM statements that do not include a THRU phrase.

Table 1 – RDz Version 9.1.2 Out-of-the-Box Code Review Rules

ALL OF THESE PARENTHESES AND SEMICOLONS TOTALLY RUIN THE FENG SHUI OF THE CODE.
SALLY LEARNS AN IMPORTANT LESSON ABOUT INVITING THE RIGHT PEOPLE TO HER CODE REVIEWS.

RAA Search Patterns – Finding what you're looking for with Rational Asset Analyzer

*This article focuses on the use of IBM's **Rational Asset Analyzer** product in doing standard z/OS Maintenance and Production Support search tasks.*

By Ravikanth Chali

Enterprise Search – "*Simple yet powerful*"

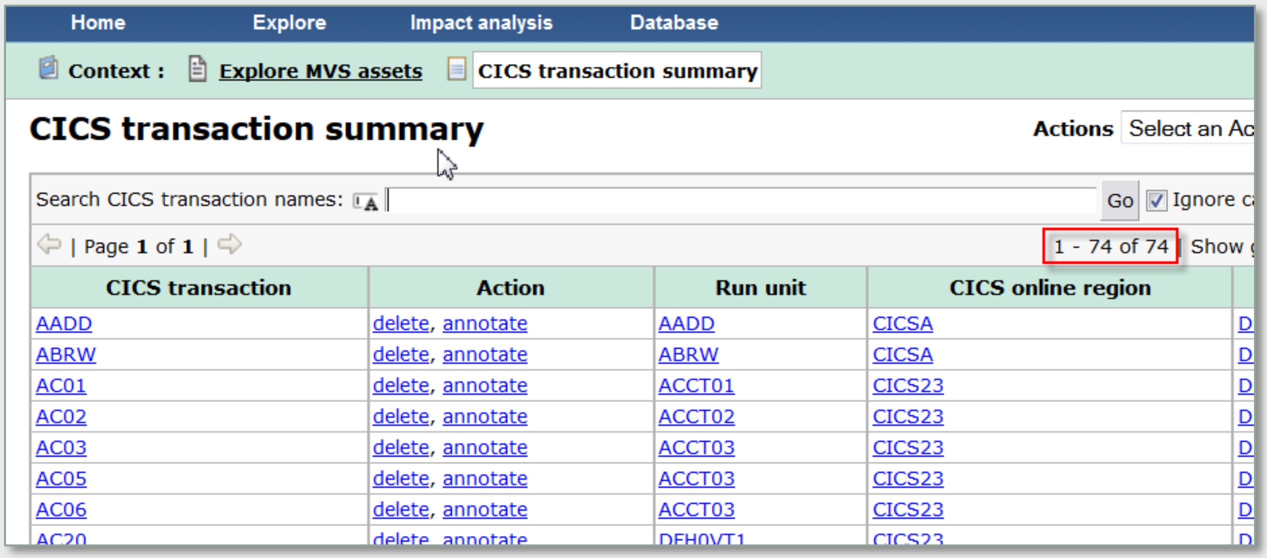
"Simple yet powerful" … How many times have you heard that axiom applied to software/technology? Mostly in the form of marketing-spin, on some new yet (seemingly) wholly un-field-tested applied science that is in search of some euphemism for; **"This will easy after you've spent 6 months using it."** I mean… stuff can be simple, and stuff can be powerful. But simple stuff generally isn't powerful. And powerful stuff generally isn't simple. I don't know about you, but these tired homilies should go the way of the CASE tools and the 4GLs of the 1980s and 1990s. (May they rest-in-peace, and we hope that Fran Tarkenton eventually made back his investment in IEF).

However, I've heard various IBM'rs describe RAA as simple yet powerful – and like the exception that proves the rule, with RAA they have a point.

Intelligent Searching – using RAA

You might think that using Rational Asset Analyzer to search for something across your inventoried Enterprise assets is simple. And, it can be simple.

For example, from MVS assets you can go directly to a list of all CICS transactions inventoried (**Figure 13**). Which works well enough with a small number of assets but what IF you've inventoried 74,000 transactions? You're not *seriously* going to page/scroll down through a list that large…….. right?



CICS transaction summary

Search CICS transaction names:

| Page **1** of **1** |

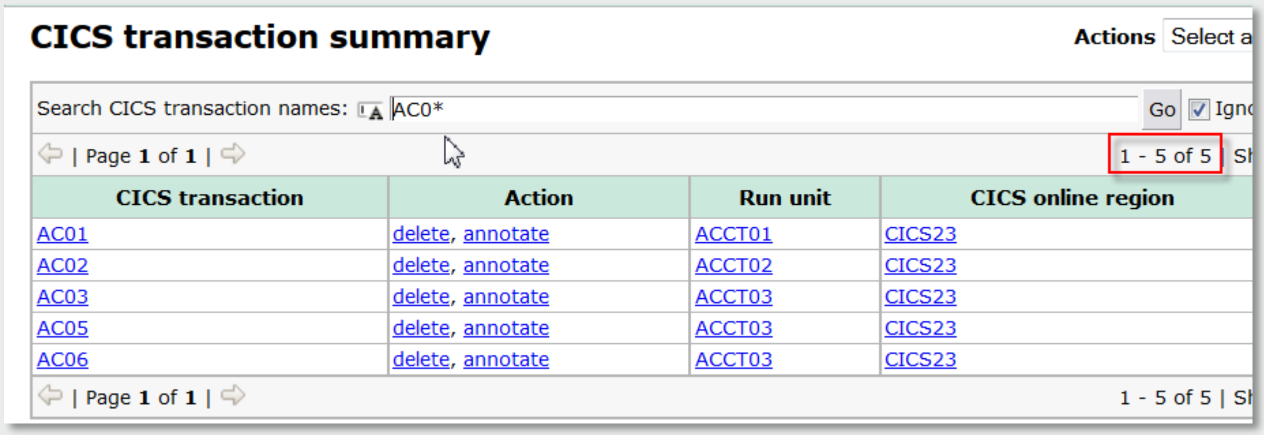
1 - 74 of 74

CICS transaction	Action	Run unit	CICS online region	
AADD	delete, annotate	AADD	CICSA	D
ABRW	delete, annotate	ABRW	CICSA	D
AC01	delete, annotate	ACCT01	CICS23	D
AC02	delete, annotate	ACCT02	CICS23	D
AC03	delete, annotate	ACCT03	CICS23	D
AC05	delete, annotate	ACCT03	CICS23	D
AC06	delete, annotate	ACCT03	CICS23	D
AC20	delete, annotate	DFH0VT1	CICS23	D

Figure 13. RAA's CICS Transaction Summary

Short answer – no… you're not. You're going to search through the RAA repository using a search pattern (see **Figures 14 and 15**).

- **Figure 14** shows the same CICS Transaction Summary list filtered with a search term of AC0* - all transactions starting with AC0.
- **Figure 15** shows a Program summary list being searched using the Advanced search functionality. Advanced search options allow you to logically OR or AND (include/exclude) subsets of the complete Enterprise inventory in your search results list



CICS transaction summary

Actions Select a

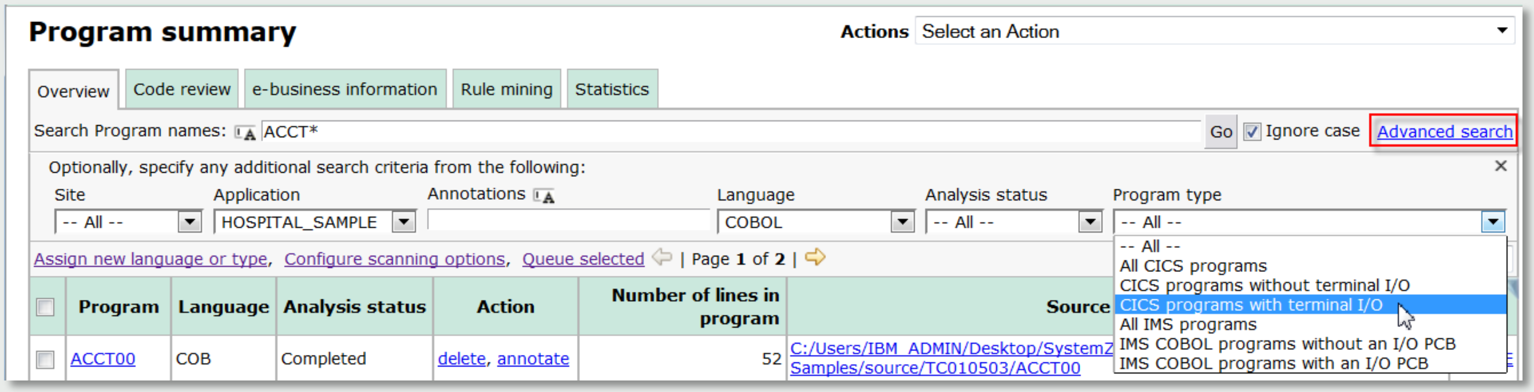
Search CICS transaction names: AC0* Go ✓ Ign

| Page **1** of **1** | 1 - 5 of 5 | Sh

CICS transaction	Action	Run unit	CICS online region
AC01	delete, annotate	ACCT01	CICS23
AC02	delete, annotate	ACCT02	CICS23
AC03	delete, annotate	ACCT03	CICS23
AC05	delete, annotate	ACCT03	CICS23
AC06	delete, annotate	ACCT03	CICS23

| Page **1** of **1** | 1 - 5 of 5 | Sh

Figure 14. RAA's CICS Transaction Summary – Filtered using Search pattern: AC0*



Program summary

Actions Select an Action

Overview | Code review | e-business information | Rule mining | Statistics

Search Program names: ACCT* Go ✓ Ignore case Advanced search

Optionally, specify any additional search criteria from the following:

Site	Application	Annotations	Language	Analysis status	Program type
-- All --	HOSPITAL_SAMPLE		COBOL	-- All --	-- All --

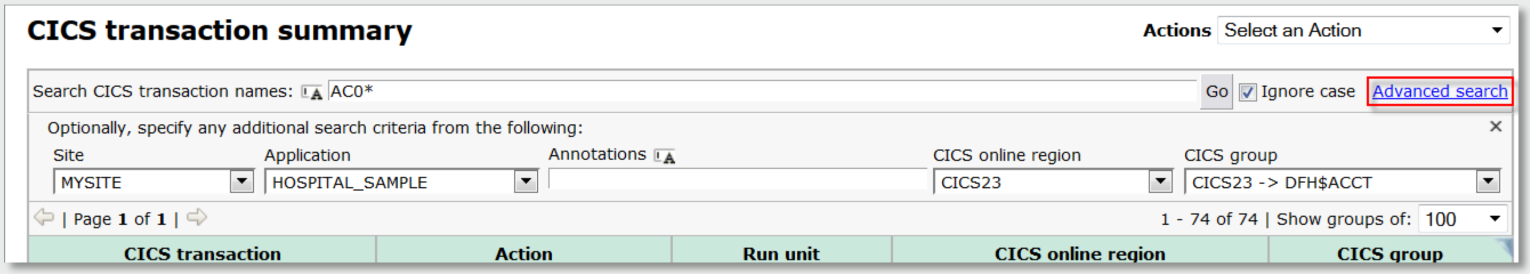
Assign new language or type, Configure scanning options, Queue selected | Page **1** of **2** |

-- All --
All CICS programs
CICS programs without terminal I/O
CICS programs with terminal I/O
All IMS programs
IMS COBOL programs without an I/O PCB
IMS COBOL programs with an I/O PCB

	Program	Language	Analysis status	Action	Number of lines in program	Source
☐	ACCT00	COB	Completed	delete, annotate	52	C:/Users/IBM_ADMIN/Desktop/SystemZ Samples/source/TC010503/ACCT00

Figure 15. RAA's Program Summary – Advanced search with different terms

Figure 15 is searching for all CICS/COBOL programs, inventoried into the **HOSPITAL_SAMPLE** Application, with names that start with: **ACCT** and that use BMS or some other form of **terminal I/O**.



CICS transaction summary

Actions Select an Action

Search CICS transaction names: AC0* Go ✓ Ignore case Advanced search

Optionally, specify any additional search criteria from the following:

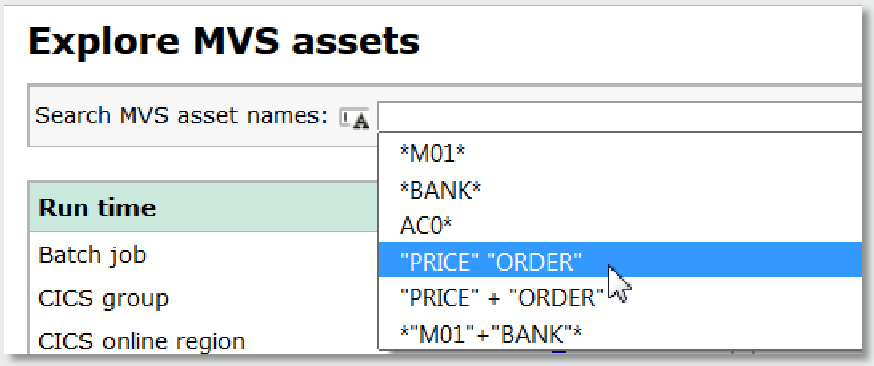
Site	Application	Annotations	CICS online region	CICS group
MYSITE	HOSPITAL_SAMPLE		CICS23	CICS23 -> DFH$ACCT

| Page **1** of **1** | 1 - 74 of 74 | Show groups of: 100

CICS transaction	Action	Run unit	CICS online region	CICS group

Figure 16. RAA's CICS transaction Summary – Advanced search with different terms

Most of the RAA summary pages provide an Advanced search capability (see **Figure 16**) – if only to subset by the Site and Application elements.



Searching through the Repository

Using RAA, you are searching for an asset's identifier in the metadata model, not merely performing a text search. This method will return fewer "false positives" or results that match a semblance of your search criteria but do not help you find what you are looking for. In fact, they complicate your work, because you need to inspect and eliminate search "hits".

Because the size, which is measured in the number of artifacts inventoried for your enterprise assets, is so enormous, the ability to perform pinpoint searching is critical. Rational Asset Analyzer can perform pinpoint searching, because it has broad, deep, and common requirement-specific searching capabilities:

- Rational Asset Analyzer has a sophisticated but not overly-complex set of search parameters and wildcard characters.
- Under Advanced Search, there are specific common requirement results and additional search features, such as looking for variables with a physical or logical length within a range or searching for unused variables.

The MVS assets model-view searching is the "simple" aspect of RAA. So let's now look at Rational Asset Analyzer's search/filtering features using wildcard text patterns.

Enterprise search – filtering results with wildcard text patterns

These examples demonstrate the rules that are used for common name searches. Square brackets ([]) are used to isolate the actual strings. The brackets are not part of the strings or phrases themselves:

- An asterisk (*) wildcard matches zero or more characters.
  - **`Example: [Z*] matches [Z] or [Z1] or [Z12]`**

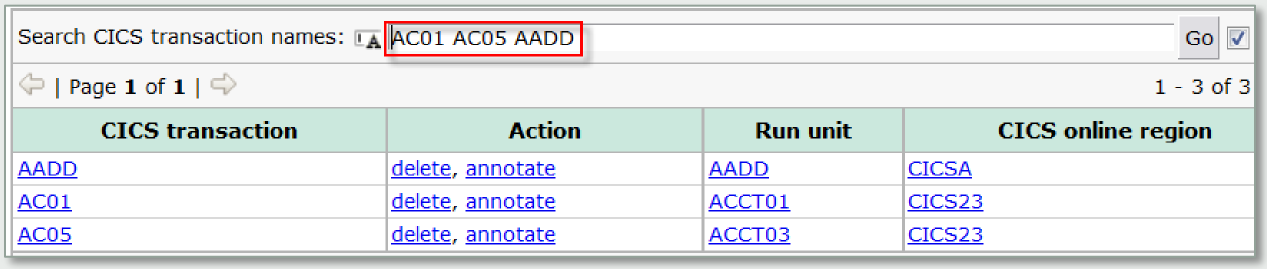
- A question mark (?) wildcard matches exactly one character and ensures that the match is of the same length. Examples:
  - **`[Z?] matches [Z1] but not [Z].`**
  - **`[Z??] matches [Z12] but not [Z] or [Z1]`**

- Single-quotation marks or double-quotation marks are required for strings that have embedded blanks.
- Examples:
  - **`["A VEHICLE"] matches the phrase [A VEHICLE]`**
  - **`["AVEHICLE"] matches the string [AVEHICLE]`**

Logical OR, Logical NOT and Logical AND

Rational Asset Analyzer supports logical AND, OR, and NOT searches. Multiple terms that are separated by white space and not enclosed within quotation marks are processed as a logical OR expression, for example:

- `[VDUB LEXUS FORD] matches [VDUB] or [LEXUS] or [FORD]`
- `["TEST 1" 'TEST 2'] matches [TEST 1] or [TEST 2]`
- `["'SOME STRING'"] matches ['SOME STRING']`



Search CICS transaction names: AC01 AC05 AADD Go

| Page 1 of 1 | 1 - 3 of 3

CICS transaction	Action	Run unit	CICS online region
AADD	delete, annotate	AADD	CICSA
AC01	delete, annotate	ACCT01	CICS23
AC05	delete, annotate	ACCT03	CICS23

Figure 17. RAA's Literal summary page searched with logical OR terms

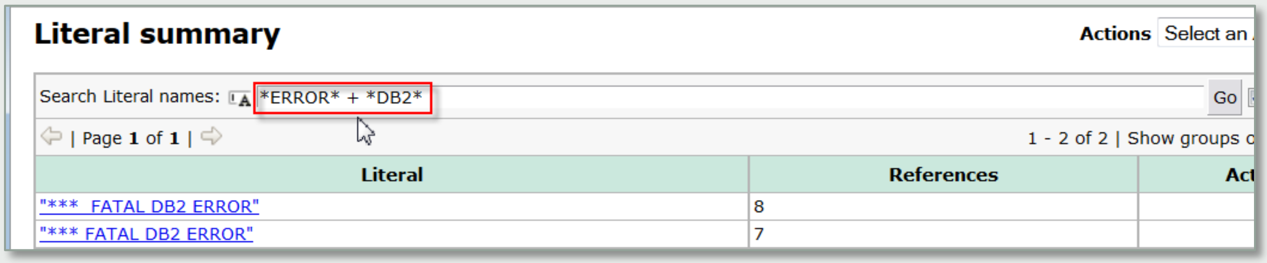
A minus (-) sign is a logical NOT operator and excludes terms. The logical NOT is handy as a false positive filter, for example,

- `[A* - *C] matches [AB] and [ABCD] but not [ABC]`

A plus (+) sign is a logical AND operator. Remaining search tokens that do not begin with + are OR operators, for example:

- `[*A* +*B*] matches [*A*] and [*B*]`
- `[+*A* +*B*] matches [*A*] and [*B*]`
- `[*A* *B* +*C*] matches (([*A*] or [*B*]) and [*C*])`
- `[*A* +*B* *C*] matches (([*A*] or [*C*]) and [*B*])`
- `[*A* *B* +*C* -*D*] matches (([*A*] or [*B*]) and [*C*] and not [*D*])`

Using logical AND search patterns for searching against simple RAA meta-data objects such as program or transaction names, batch job names, tables, files, etc. is not going to net you much. Use logical AND patterns against complex meta-data such as literal strings.



Literal summary Actions Select an

Search Literal names: *ERROR* + *DB2* Go

| Page 1 of 1 | 1 - 2 of 2 | Show groups o

Literal	References	Act
"*** FATAL DB2 ERROR"	8	
"*** FATAL DB2 ERROR"	7	

Figure 18. RAA's Literal summary page searched with logical AND terms

Note that, from **Figure 18** – it wouldn't matter if you coded the **DB2** or the **ERROR** search text first.

Search patterns that include the escape character

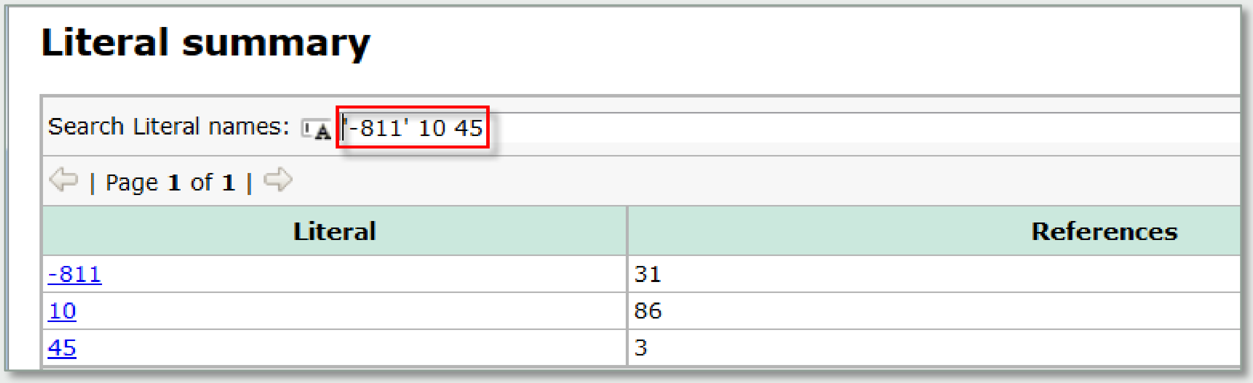
The backslash character (\) is an escape character for the asterisk (*), question mark (?), plus sign (+), minus sign (-), single quotation mark ('), double quotation marks ("), and backslash (\), for example:

- **`[*\**] matches any string with the literal * in it`**
- **`[+A* +*\+*] matches all strings that start with A and that contain +`**

Advanced search patterns: Numeric range searching

For advanced search value fields that are numeric, such as literals, or variable searches - for fields with separate physical or logical lengths, you can enter multiple values and ranges separated by blanks, for example:

- **[10 20] matches 10 or 20**
- **[<3 >10] matches less than 3 or greater than 10**
- **[<=3 >11] matches (less than or equal to 3) or greater than 11**
- **[<3 >=11] matches less than 3 or (greater than or equal to 11)**
- **[1-4 45] matches (between 1 and 4) or 45**



Literal summary

Search Literal names: '-811' 10 45

| Page 1 of 1 |

Literal	References
-811	31
10	86
45	3

Figure 18. RAA's Literal summary page, searched with numeric values

Note from **Figure 18** that searching on negative numbers requires enclosing terms in single quotes.

Summary

Often the process of analysis begins by identifying the variation in the names that are used to refer to the component that needs to change. This situation is especially true of data elements. Because naming standards are not always followed, you must discover what variations have been used. For example, part number might be written as PARTNO, PART-ID, PART-NO, PART-NUM, PTNBR, PT-NO, and so on.

A simple text scanning tool, even when combined with a general-purpose editor, such as ISPF, requires multiple searches, one search for each variation. Why? Because if you use partial names, such as 'PART', a simple text scan will also return matches to strings, such as PARTNER and PARTIAL. These unwanted matches, which are known as false positives need to be assessed and eliminated, which is typically a manual operation. With the size and scope of production systems where the number of variables can easily reach tens of millions, this elimination is an onerous prospect.

Our view…

RAA's Repository can house the meta-data for your all of your Enterprise applications. It can also house a subset of your code. Either way intelligent searches against your codebase – where you've localized the results you're looking for will save both time & money. This article focused on different search pattern terms and wildcard characters. Future articles will continue to drive through aspects of this tool that can lighten your workload and simplify your daily tasks.

Next steps…

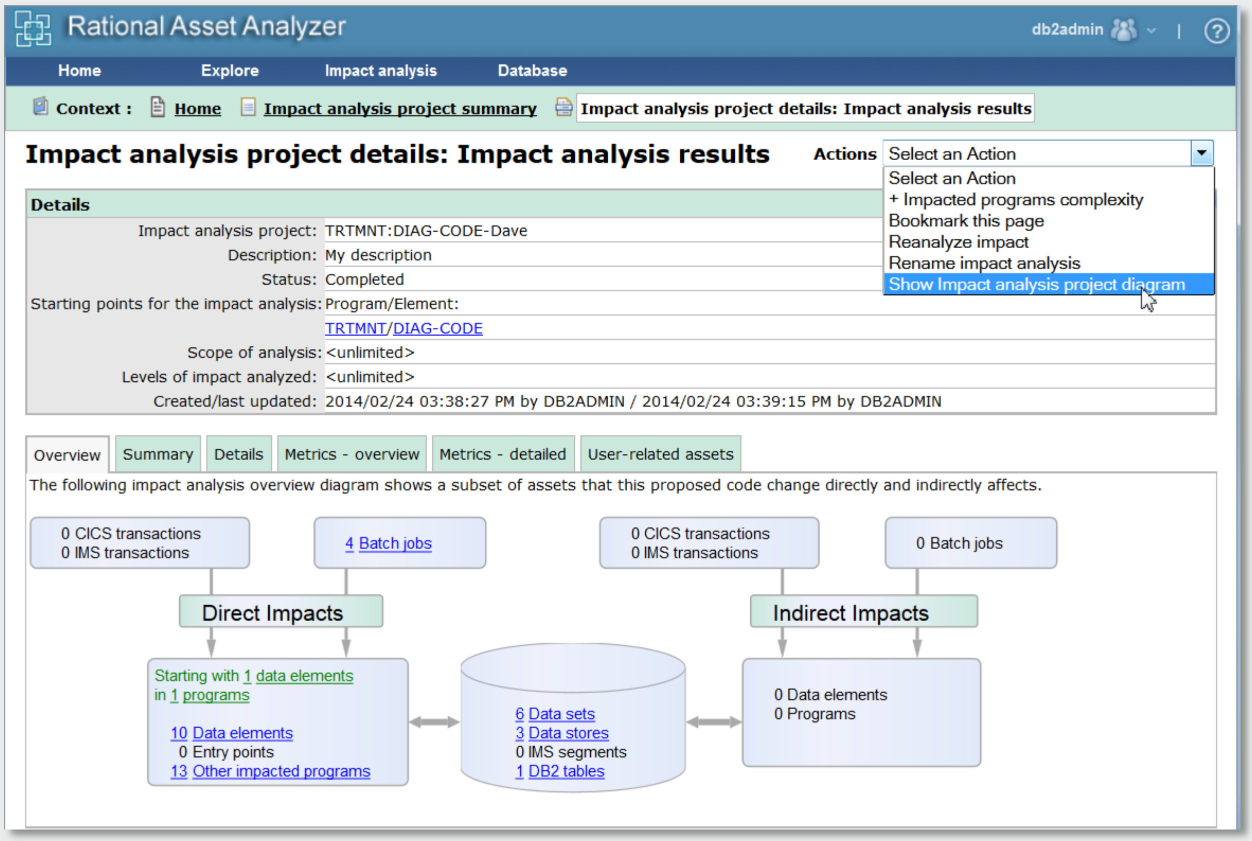
Where can you learn more about RAA…either from us or from IBM. IBM has also published a few RAA Redbooks with RAA content—such as: http://www.redbooks.ibm.com/abstracts/SG247868.html?Open

At Royal Cyber we also do **RAA demonstrations** - for those of you who might want to see RAA running vs. a few static screen captures (and really… there's no comparison, static captures/Redbooks do **not** do RAA justice)

We also consult on RAA. We bring a 3rd Party objectivity to your projects and our work on them. And we have the practical/production expertise to help you manage true production projects. Finally, we offer high-quality **RAA training** that was built from our customer and personal experience with RAA.

Contact us about our RAA service offerings Feel free to request an online/remote demo.

Ravikanth Chali is a senior technical consultant with Royal Cyber. He has ten years of experience in Mainframe application/ tools development including Eclipse and RDz plugin-ins. Ravi is very well versed in the usage and best practices related to RDz and RAA. He has world-class and unique specialist skills writing Eclipse plug-ins to RDz.



Impact analysis project details: Impact analysis results

Details

Impact analysis project:	TRTMNT:DIAG-CODE-Dave
Description:	My description
Status:	Completed
Starting points for the impact analysis:	Program/Element:
	TRTMNT/DIAG-CODE
Scope of analysis:	<unlimited>
Levels of impact analyzed:	<unlimited>
Created/last updated:	2014/02/24 03:38:27 PM by DB2ADMIN / 2014/02/24 03:39:15 PM by DB2ADMIN

The following impact analysis overview diagram shows a subset of assets that this proposed code change directly and indirectly affects.

Royal Cyber in the News

In February & March we delivered our first two IBM RDz Distance Learning classes – to over 230 new RDz users world-wide. As a first-time transition from IBM, which had been running Distance Learning for six years there were some hiccups with the logistics, but he results were as you can see below. By the time this Newsletter goes to print there will (likely) **not** be availability left in the April Distance Learning session, but you can sign up for May (through the end of 2014) sessions here: http://royalcyber.com/royal-cyber-rdz-distance-learning-training-schedule/

Rational EM News

- For details on the Royal Cyber RDz Rollout and project/task usage modeling contact us: inquires@royalcyber.com

- Details on RDz technical services: http://www.royalcyber.com/wp-content/uploads/2014/You-RDz-and-Royal-Cyber-v4.pdf

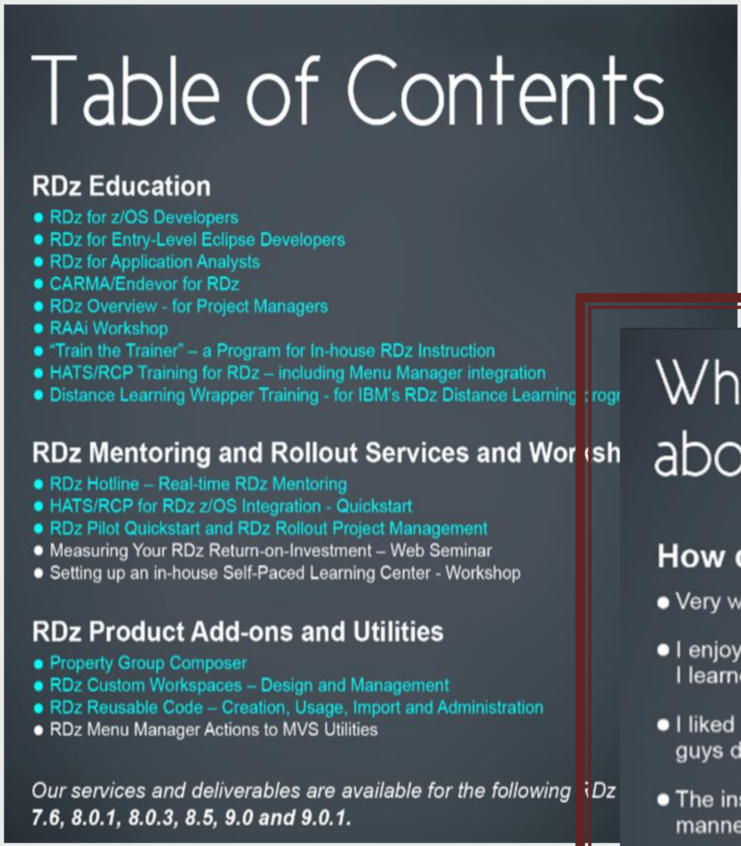


Table of Contents

RDz Education

- RDz for z/OS Developers
- RDz for Entry-Level Eclipse Developers
- RDz for Application Analysts
- CARMA/Endevor for RDz
- RDz Overview - for Project Managers
- RAAi Workshop
- "Train the Trainer" – a Program for In-house RDz Instruction
- HATS/RCP Training for RDz – including Menu Manager integration
- Distance Learning Wrapper Training - for IBM's RDz Distance Learning rogr

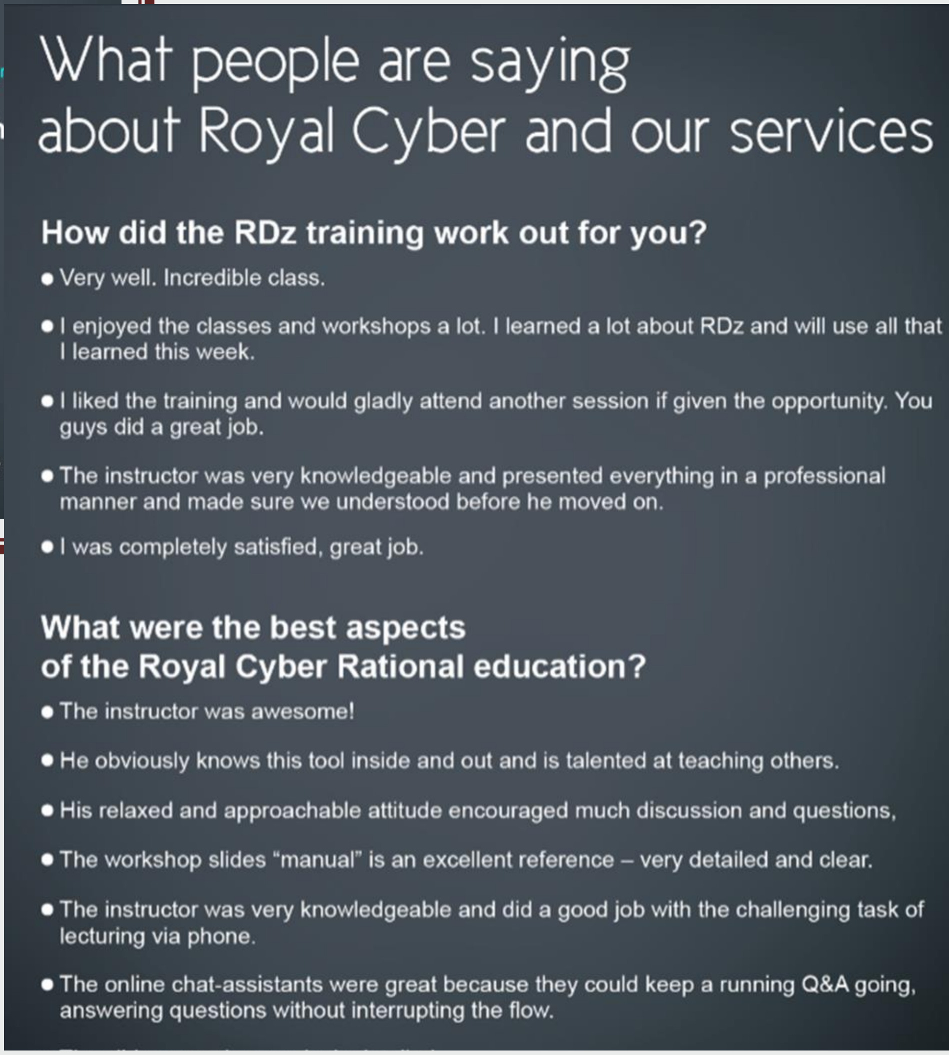
RDz Mentoring and Rollout Services and Wor sh

- RDz Hotline – Real-time RDz Mentoring
- HATS/RCP for RDz z/OS Integration - Quickstart
- RDz Pilot Quickstart and RDz Rollout Project Management
- Measuring Your RDz Return-on-Investment – Web Seminar
- Setting up an in-house Self-Paced Learning Center - Workshop

RDz Product Add-ons and Utilities

- Property Group Composer
- RDz Custom Workspaces – Design and Management
- RDz Reusable Code – Creation, Usage, Import and Administration
- RDz Menu Manager Actions to MVS Utilities

Our services and deliverables are available for the following Dz
7.6, 8.0.1, 8.0.3, 8.5, 9.0 and 9.0.1.



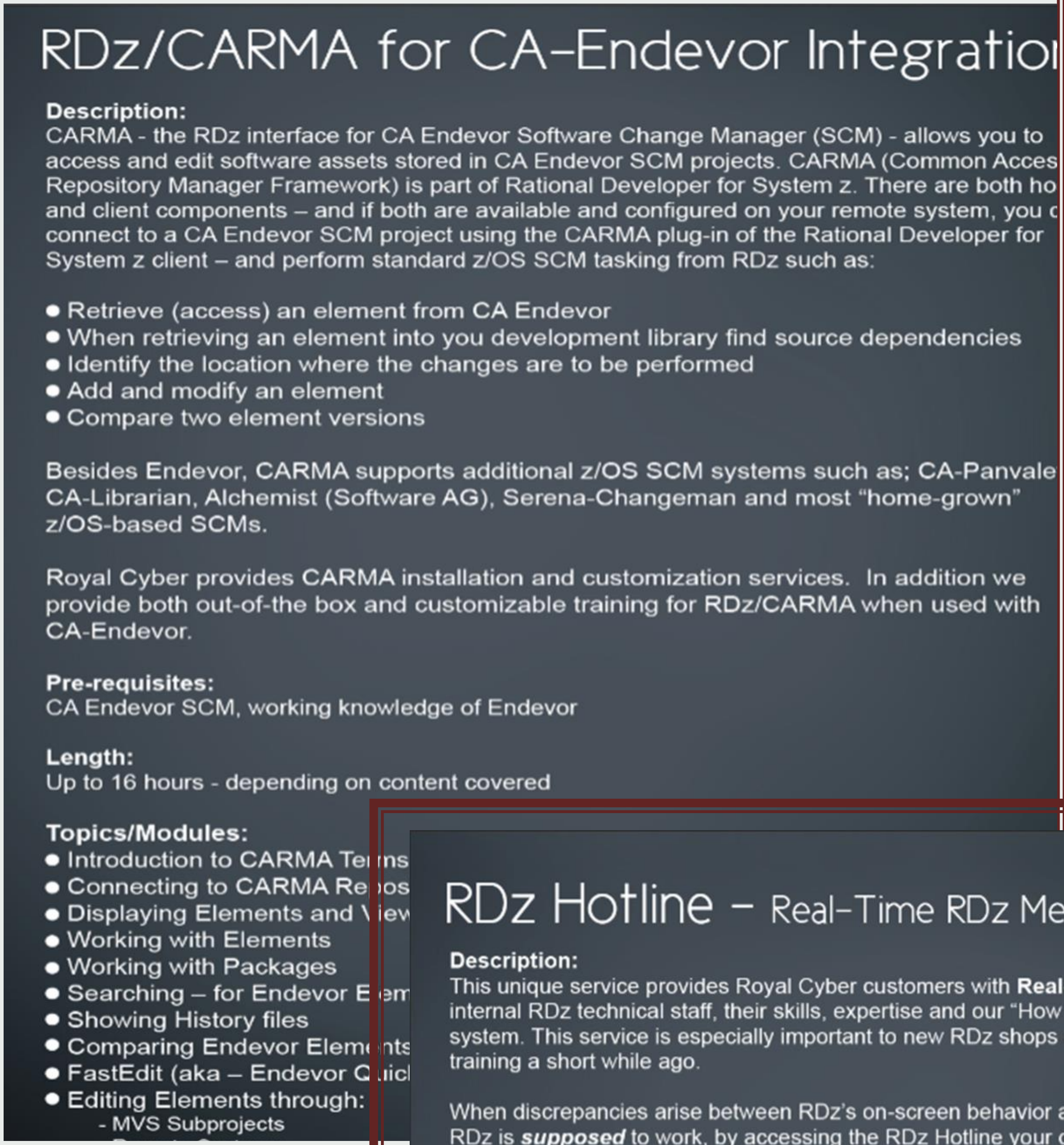
What people are saying about Royal Cyber and our services

How did the RDz training work out for you?

- Very well. Incredible class.
- I enjoyed the classes and workshops a lot. I learned a lot about RDz and will use all that I learned this week.
- I liked the training and would gladly attend another session if given the opportunity. You guys did a great job.
- The instructor was very knowledgeable and presented everything in a professional manner and made sure we understood before he moved on.
- I was completely satisfied, great job.

What were the best aspects of the Royal Cyber Rational education?

- The instructor was awesome!
- He obviously knows this tool inside and out and is talented at teaching others.
- His relaxed and approachable attitude encouraged much discussion and questions,
- The workshop slides "manual" is an excellent reference – very detailed and clear.
- The instructor was very knowledgeable and did a good job with the challenging task of lecturing via phone.
- The online chat-assistants were great because they could keep a running Q&A going, answering questions without interrupting the flow.



RDz/CARMA for CA–Endevor Integratio

Description:
CARMA - the RDz interface for CA Endevor Software Change Manager (SCM) - allows you to access and edit software assets stored in CA Endevor SCM projects. CARMA (Common Acces Repository Manager Framework) is part of Rational Developer for System z. There are both ho and client components – and if both are available and configured on your remote system, you c connect to a CA Endevor SCM project using the CARMA plug-in of the Rational Developer for System z client – and perform standard z/OS SCM tasking from RDz such as:

- Retrieve (access) an element from CA Endevor
- When retrieving an element into you development library find source dependencies
- Identify the location where the changes are to be performed
- Add and modify an element
- Compare two element versions

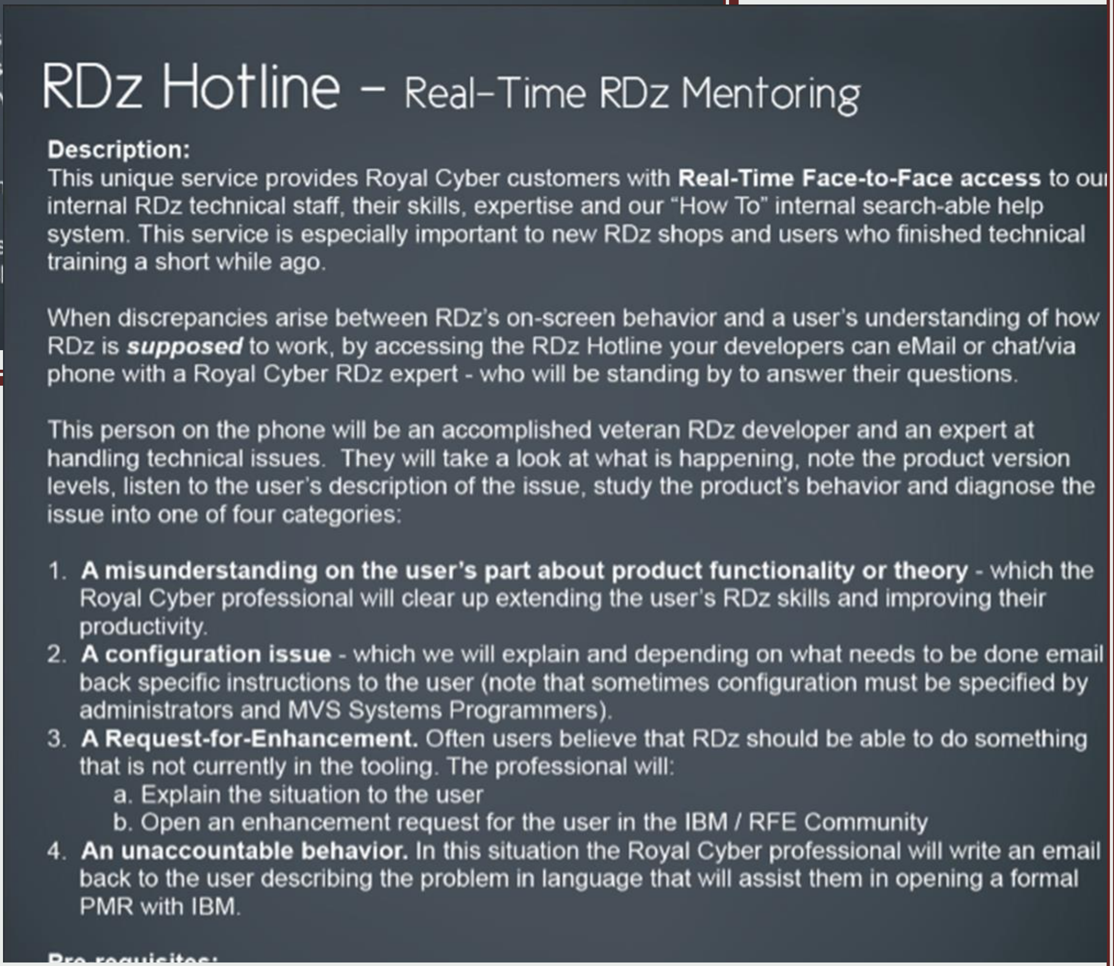
Besides Endevor, CARMA supports additional z/OS SCM systems such as; CA-Panvale CA-Librarian, Alchemist (Software AG), Serena-Changeman and most "home-grown" z/OS-based SCMs.

Royal Cyber provides CARMA installation and customization services. In addition we provide both out-of-the box and customizable training for RDz/CARMA when used with CA-Endevor.

Pre-requisites:
CA Endevor SCM, working knowledge of Endevor

Length:
Up to 16 hours - depending on content covered

Topics/Modules:
- Introduction to CARMA Te
- Connecting to CARMA Re
- Displaying Elements and V
- Working with Elements
- Working with Packages
- Searching – for Endevor E
- Showing History files
- Comparing Endevor Eleme
- FastEdit (aka – Endevor Q
- Editing Elements through:
  - MVS Subprojects



RDz Hotline – Real-Time RDz Mentoring

Description:
This unique service provides Royal Cyber customers with **Real-Time Face-to-Face access** to ou internal RDz technical staff, their skills, expertise and our "How To" internal search-able help system. This service is especially important to new RDz shops and users who finished technical training a short while ago.

When discrepancies arise between RDz's on-screen behavior and a user's understanding of how RDz is ***supposed*** to work, by accessing the RDz Hotline your developers can eMail or chat/via phone with a Royal Cyber RDz expert - who will be standing by to answer their questions.

This person on the phone will be an accomplished veteran RDz developer and an expert at handling technical issues. They will take a look at what is happening, note the product version levels, listen to the user's description of the issue, study the product's behavior and diagnose the issue into one of four categories:

1. **A misunderstanding on the user's part about product functionality or theory** - which the Royal Cyber professional will clear up extending the user's RDz skills and improving their productivity.
2. **A configuration issue** - which we will explain and depending on what needs to be done email back specific instructions to the user (note that sometimes configuration must be specified by administrators and MVS Systems Programmers).
3. **A Request-for-Enhancement.** Often users believe that RDz should be able to do something that is not currently in the tooling. The professional will:
   a. Explain the situation to the user
   b. Open an enhancement request for the user in the IBM / RFE Community
4. **An unaccountable behavior.** In this situation the Royal Cyber professional will write an email back to the user describing the problem in language that will assist them in opening a formal PMR with IBM.

Pre-requisites: