RC • Royal Cyber
& • Development Modernization
You • Newsletter

"Technical Content For A Rational World"

Volume 1 – Issue #3

In this issue:

Recycled Code – RDz's Reusable Code Features

This article explores several features of IBM's Rational Developer for System z applied to the problem space and value of "code reuse"

By Chris Leland, Royal Cyber

There have been many, many, many, many attempts throughout the history of software development to provide frameworks, languages, and tools that assist programmers in building applications from reusable parts – usually tested/production-ready code. Reuse is a linchpin of Object Orientation (the Inheritance construct) – and it's hard in theory to argue against it. So we won't.

However, COBOL/Assembler/PL1 aren't – in general practice O-O languages. Please, please, ***please*** don't drag O-O COBOL onto the dance floor. We've tangoed with it– and when you get up close ... when you actually study an O-O COBOL program? It resembles an off-label Botox treatment, applied to an aging Hollywood actor.

Anyway… so, if no Inheritance, then what's the value of code reuse? Actually two things? 1. Common code functionality and 2. complex "plumbing" functionality. We're betting that both of these are currently implemented in your production systems via COPY or INCLUDE statements – with some "advanced reuse" provided by "COPY REPLACING". These are effective, traditional options that; save time, cut down on systems-development errors (both build and run-time errors) and standardize your code base.

There's an unofficial but common 3rd option for code reuse which is the ISPF command line COPY <External Source File> command. Often developers utilize command line copy to snare fragments, bits of code – even sometimes entire program or file skeletons. This practice is widespread, but would be enhanced by more copy-into flexibility (i.e. the means to implement the paste operation substituting strings or variable values). It's also done at a block level – in other words line(s) at a time are copied into your program – you can't copy within a statement. It is this ISPF command line COPY functionality that RDz's reusable code features most resembles. Except the two options (Snippets and Templates) surmount the limitations of ISPF's command line COPY.

Code Snippets

Code Snippets are fragments of code, typically in some programming language (COBOL, PL/I, Assembler, JCL, CLIST, REXX, C/C++, Java, SQL, MFS, CA-Ezytrieve, SAS, etc.) that are inserted into a file. The file would most likely be a program – ostensibly the same language as the Snippet – but there are relatively few exceptions / limitations to Code Snippet use. So in theory you could use Snippets as building blocks for virtually any plain-text z/OS software resource.

You create a snippet by:

1. Copying a select of code
2. Right-clicking over the appropriate "palette/drawer" in the Snippets view (see Figure 1)
3. And selecting Paste as Snippet…

In the COBOL/PL1 Editors an "Add to Snippets…" option exists off of the Context Menu – within edit.

The use or consumption of an existing Snippet is just as straightforward.

From within an LPEX, COBOL or PL/I editor session:

1. Place your cursor at the exact focal point (position in the source) where you want a code snippet inserted;
    - This includes at the beginning of a line or anywhere inside the line or statement.
2. Locate the Code Snippet within the Snippets view, and inside the Snippet palette/drawer (see **Figure 1**)
3. Double-Click the Snippet

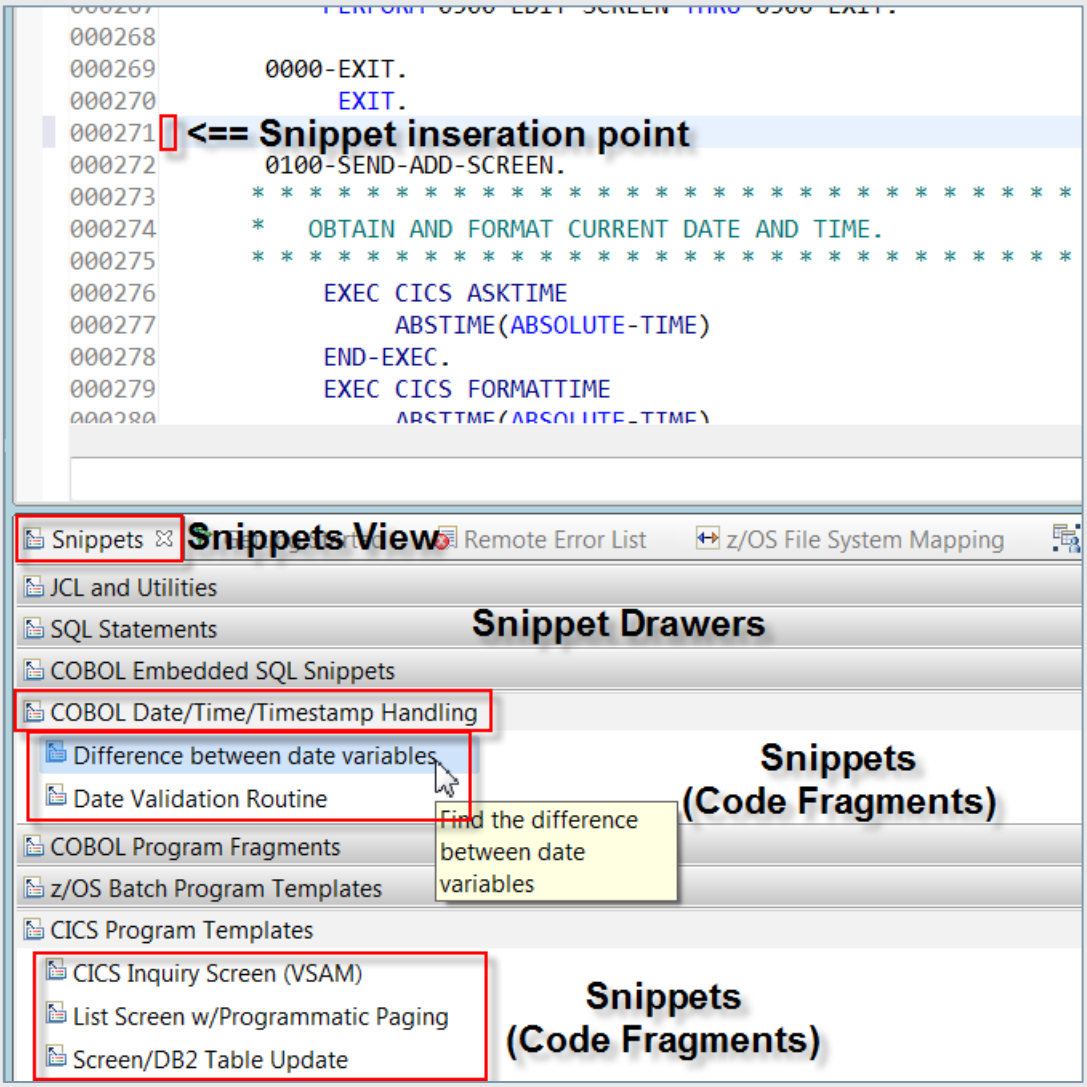


Figure 1 – Snippet Palette/Drawers within the Snippets View – and setting the Snippet insertion point

The code in the Snippet will be added into your file at the insertion point (see **Figure 1**) – either verbatim, or if there are variables in the Snippet you can:

- **Accept default values for the variable**
- **Over-ride the values before the code is inserted**

What are these Snippet *Variables*?

Snippet Variables are placeholders in the Snippet artifact where the user is prompted for a value, which is substituted into the Snippet at a defined location. From **Figure 2** you can see that there's a JCL Job Card Snippet which contains a standard z/OS JOB statement. The Snippet developer has built two variables into the Snippet: JobName and MsgClass. When the user opens a JCL file, inserts his cursor on top (blank) line in the file and double-clicks the JobCard Snippet, the Snippet code will be inserted into the JCL. But first the user is prompted for JobName and MsgClass. Whatever the user enters into the values table for JobName and MsgClass will be added into the final Snippet code (again, see **Figure 2**). So it's this "variables" feature of RDz's Snippets that provides more flexibility, functionality and in general more choices and options for reuse. A technical (best practice) note that if your snippet begins with a new line of code, begin your snippet with a blank line. This ensures that the inserted code begins at the correct column position.

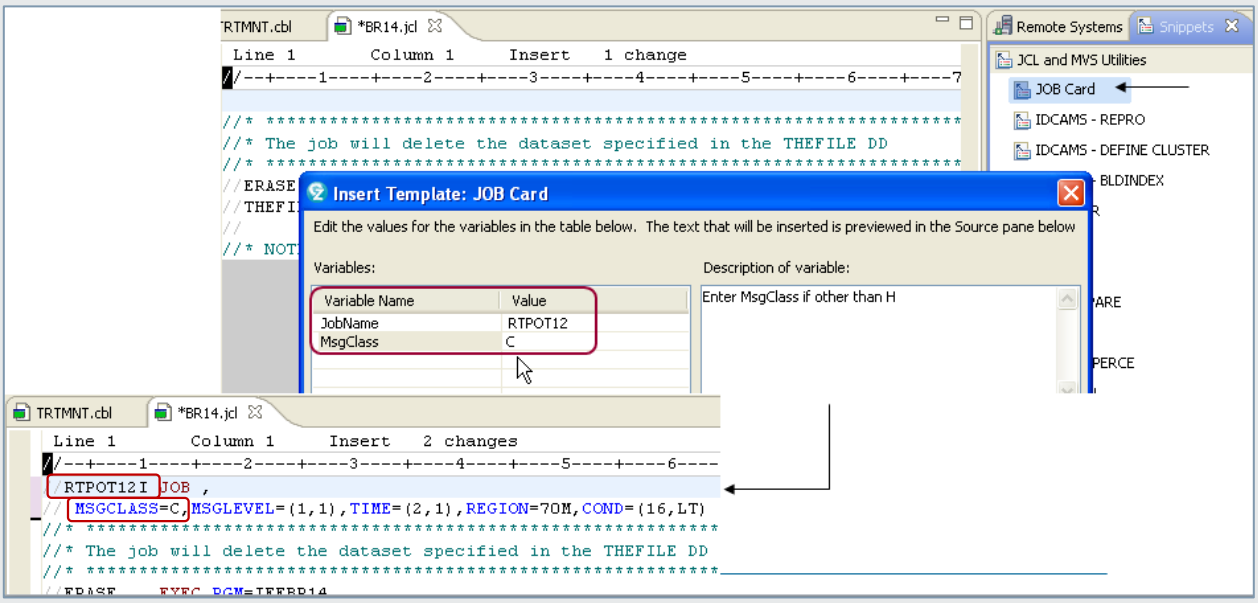


Figure 2 – Variables substituted for JobName and MsgClass in a JCL Code Snippet

Code Templates in RDz

An alternative to Snippets are RDz "Code Templates". Code Templates are pieces of reusable code that are defined in RDz's **Preferences** as shown in **Figure 3 – RDz's COBOL Templates**).

Code Templates are (to a degree) language sensitive, and available for; Assembler, PL/I, JCL, COBOL, and SQL. They can be used for small/granular development components such as single statements or keywords (that's what's shipped with RDz).

Or Code Templates could potentially be defined for:

- **COBOL,**
  - **Statement combinations and common code –** such as logging, audit or ABEND routines
  - **Paragraphs/sections**
  - **Embedded SQL Cursor routines**
  - **Entire programs**

- Template functionality is also shipped for other RDz-supported language elements, features and structures:
  - **PL/I**
  - **Assembler**
  - **SQL**
  - **Java**
  - **C/C++**
  - **JCL** – JCL keywords are shipped with RDz, but you can add to the JCL functionality by creating Templates for:
    - **Jobcard**
    - **Parms**
    - **Common DD cards**
    - **MVS Utilities**
    - Etc.

By "language sensitive" what I mean is that the RDz tooling provides Context (see Figure 3) for the definition of Templates. And what this ("context") means is that you can specify that certain Templates will only be proposed in certain contexts; SQL code within embedded EXEC SQL statements, procedural statements in the PROCEDURE DIVISION, etc.

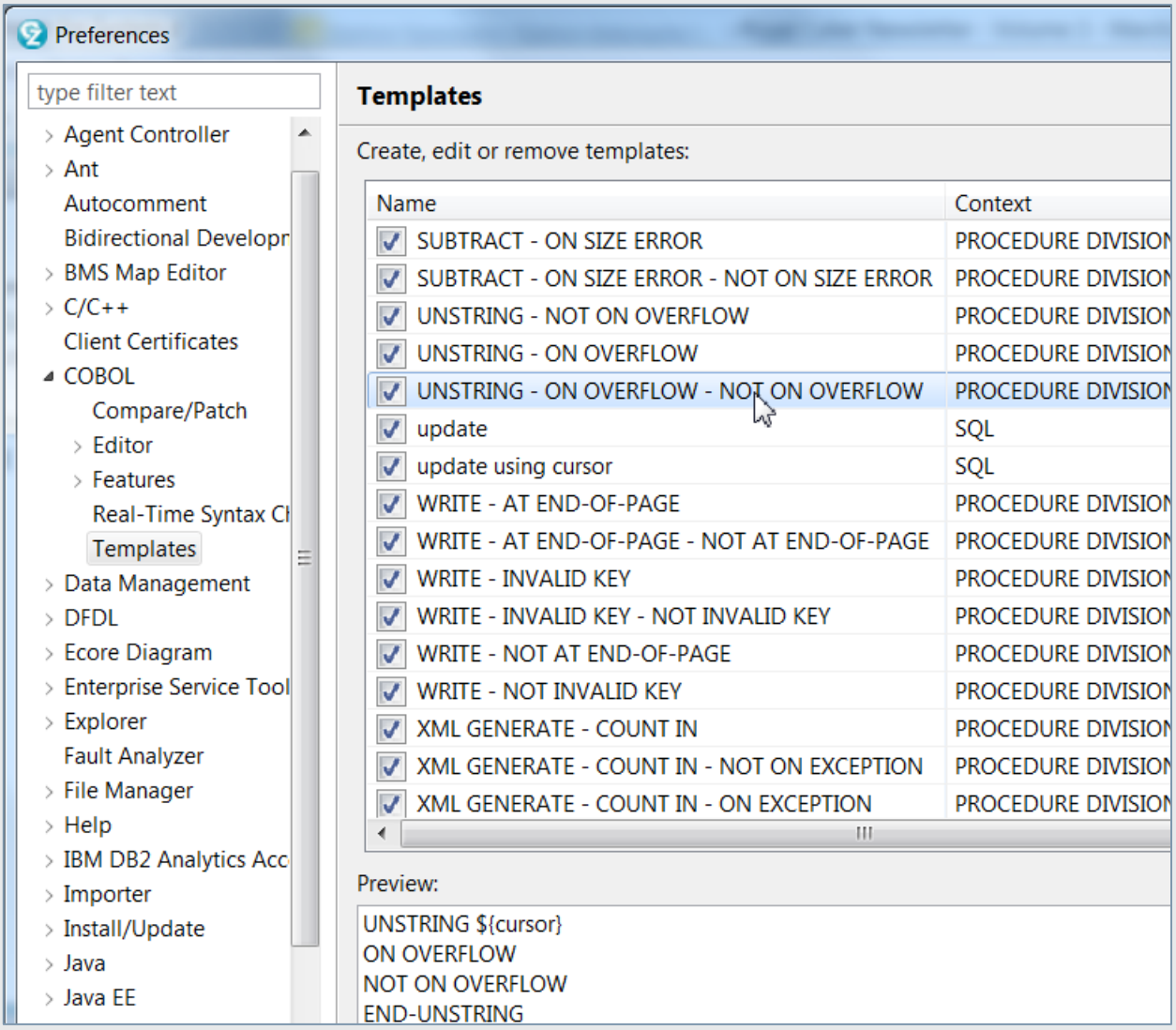


Figure 3 – RDz's default COBOL Code Templates – defined in Preferences/shipped with the product

Using Code Templates

To use an existing Code Template you press the "Content Assist" hot-key combination (Ctrl+Spacebar) from inside of an RDz edit session (see **Figures 4a, 4b and 4c**). This process works as follows:

1. Place your cursor (focal-point) in the code where you wish to use a Code Template
2. Press **Ctrl+Spacebar** – a list of proposals valid within your coding context pops up (see Figures **4a, 4b** and **4c**)
3. Scroll through the list to find the Template you want. Mouse-over to get any Comments that were added into the Template
4. Double-click to select and insert the Template code into your source file

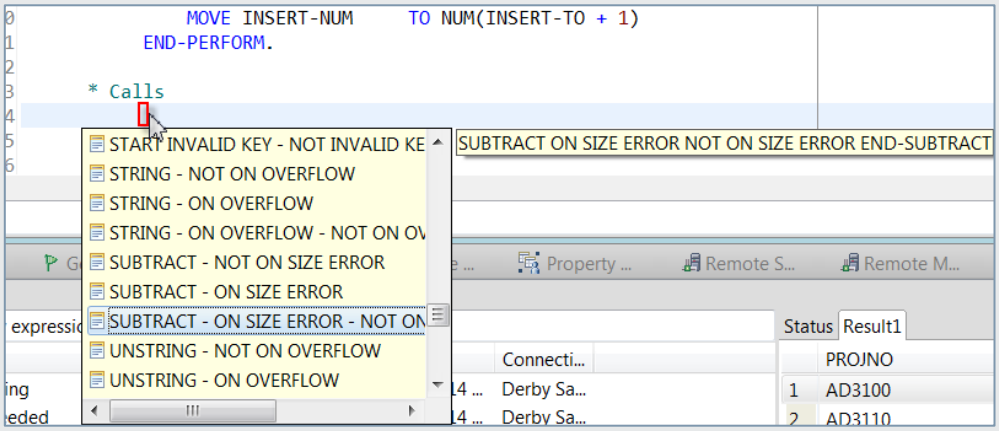


Figure 4a – Inserting a COBOL PROCEDURE DIVISION statement from a Code Template

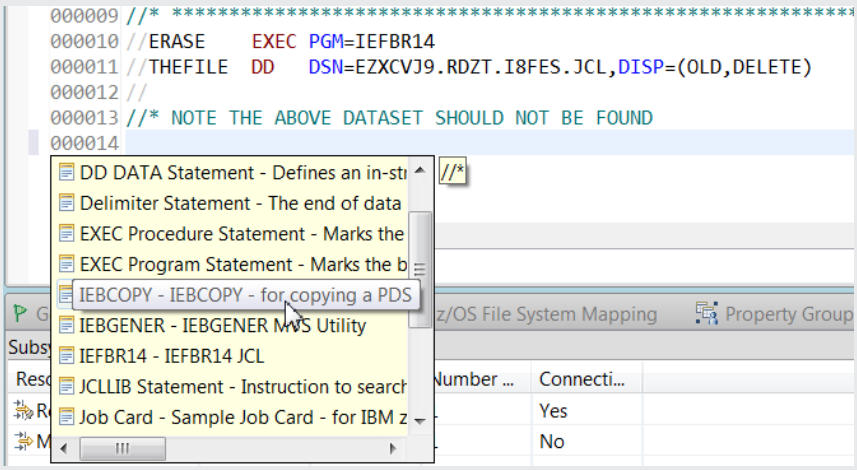


Figure 4b – Inserting an entire IEBCOPY Utility from a JCL Code Template

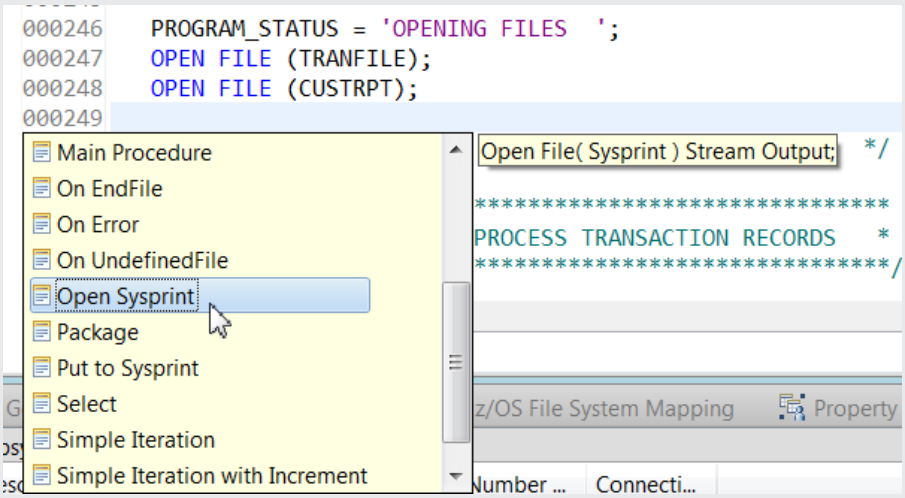


Figure 4c – Adding an Open Sysprint statement from a PL/I Code Template

Defining Code Templates

Figure 3 shows some of the default Templates in COBOL for RDz v9.0.1. You can modify the default Code Templates; selecting and deleting ones you don't like, editing/saving/modifying existing Templates. And you can extend/augment and modify the list as follows:

- From Window > Preferences, expand COBOL >
- Select Templates and click **New**…
- From New Template specify Template name, Context, (optional) Description and code Pattern
- Click OK – and your new code Template is ready to activate using **Ctrl+Spacebar**

Variables in Code Templates

You may notice that there are Variables available to insert into RDz Code Templates. For the most part, these variables are not flexible/open-ended like those in RDz's **Code Snippets**, so we rarely see them used.

However, there is one really nice variables option; **cursor** – which forces a cursor stop/point wherever you place it. This stop/point allows you to press subsequent **Ctrl+Spacebar** actions, to bring program variables/paragraphs/section names/etc. into the Template-completed statement or pattern.

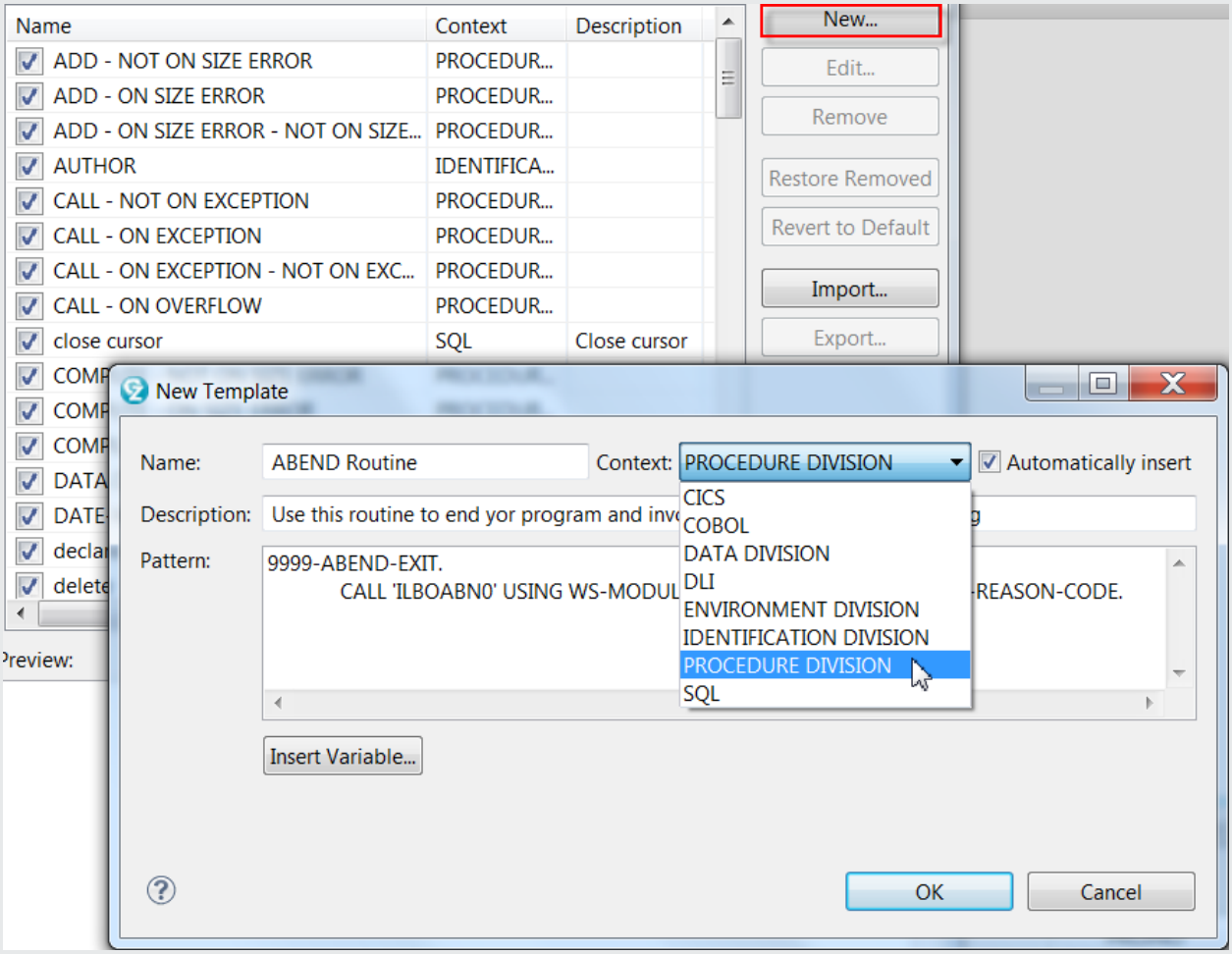


Figure 5 – Define a new COBOL PROCEDURE DIVISION Template

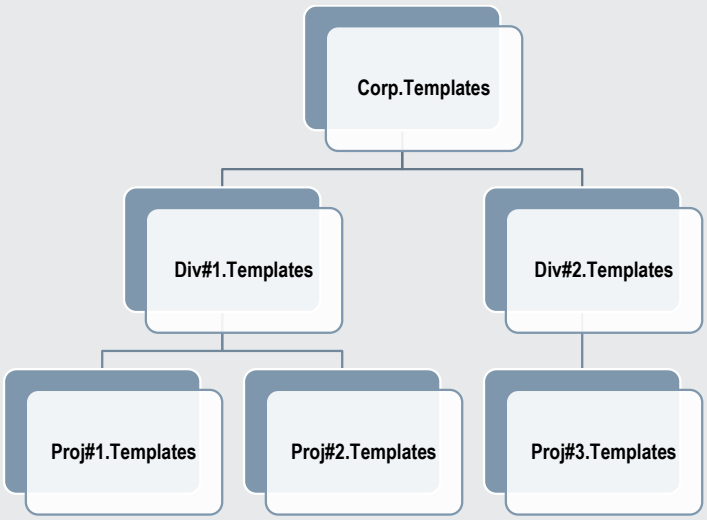
Managing and Administering Code Snippets and Code Templates

Both Code Snippets and Code Templates are Workspace-managed artifacts (they existing inside of the RDz Workspace you launch the product with).

Both Snippets and Templates can be shared among developer/development teams through standard Eclipse Export/Import actions. The unit of Code Snippet Export/Import is the palette/drawer and the unit of Code Template Export/Import is the number of selected Templates in the Preferences dialog (you can select multiple Templates for Export by holding the Ctrl or Shift key).

However, Code Templates further allows you to manage/administer Code Templates using RDz's "Push-to-Client" technology. This feature provides z/OS LPAR-hosted (single-site) management of updates to RDz dependent artifacts; Workspaces and Workspace components such as Templates other Preferences, etc.

Best Practices dictate that a few dedicated developers setup and manage Snippet/Template content in a hierarchical distribution fashion for maximum ROI results – something like the design shown **in Figure 6.**



.

Figure 6 – Hierarchical Snippet/Template Model/Design

Our Take … Code Snippets vs. Templates

Whether you should use Snippets or Templates is largely a matter of development culture – and how you expose users to these powerful options via training. It should be noted that Code Snippets and Templates are not incompatible (see **Table 1**). You can use both in your RDz development environment – and some shops do just that.

The same shops/teams/individuals that don't utilize ISPF command line COPY, probably won't use Snippets or Templates. But if you find value in / or use ISPF COPY consider taking a look at RDz Snippets and Templates.

Royal Cyber maintains robust, organized collections of both custom Snippets and Templates for our own in-house development engagements. You can obtain a copy of them (they're free!) by contacting us: http://royalcyber.com/contact-information/

Option	Snippets	Templates
Content Assist enabled?	N	Y
Block (lines) of code inserted	Y	Y
Insertion into individual lines/statements	Y	Y
Provides variable substitution for greater reuse capability	Y	N
Can be managed via Push-to-Client	N	Y
Can be Exported/Imported among workspaces	Y	Y
Can be organized into higher-level abstraction categories	Y	N
Language sensitive	N	Y
Definitions accessed via an Eclipse view	Y	N

Table 1 – Snippets vs. Templates Functional Comparison

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.

Online Transaction Static Analysis – Done Right

*This article focuses on the use of IBM's **Rational Asset Analyzer** product in CICS and/or IMS TM transaction analysis.*

By Ravikanth Chali

You've just been assigned to a new project team … or? You went to work for a new company… or? You've jumped back into some pool of complex CICS or IMS TM (you remember it as: **IMS DC**) transaction you once authored programs for – either because you have to pick up support or have to analyze a change. What's the first thing you do?

1. **Start reading the code** – either in ISPF or from listings (Dude… listings? Really? Why stop there. Perhaps there's a deck of punch cards lying around…)
2. **Phone a friend** – sorry, this isn't "Who wants to be a millionaire" – you **are** the Subject Matter Expert – no dodging this one
3. **Scramble to get your resume up on www.monster.com**

If none of the above options appeal to you - or they're just not viable the current state of software jobs being what it is, allow me to introduce you to Rational Asset Analyzer – IBM's enterprise static code analysis tooling that will provide a 4th option for tasks such as this: ***Study the semantics of the system top down, following graphic flows and using hyperlinks for navigation.*** Sound appealing? I thought so. Let's explore…

What is Rational Asset Analyzer?

Rational Asset Analyzer: sometimes called Asset Analyzer or RAA for short – is a classic static code analysis product that was created in response to the Y2K century/date math/Impact Analysis conundrum, circa 1998.

There were about a dozen such tools on the market back then, most of which have gone the way of the

IBM purchased RAA shortly into the 21st century and they've been enhancing it since then – making the product stronger, bigger (more of a z/OS enterprise-quality tool) and injecting the code base with new functionality – including (but not limited to):

- Java/J2EE analysis
- Web Services analysis
- Business Rule Mining
- Code Quality/Code Review

Perhaps in some future Royal Cyber Newsletter(s) we'll delve into these four topics (let us know what YOU are interested in) – but let's get back to analyzing online transactions. BTW - **Static Code Analysis** is the investigation of application behavior and synthesis of program semantics by reading source code - without running or debugging programs.

How RAA works…

The way you utilize RAA is that, after you've installed the RAA server (either on a Wintel server or on z/OS) – you "collect inventory" – by:

- Kicking off a wizard from the RAA front-end (an I.E. / Firefox or Chrome browser at the release and level supported by the version of RAA you've installed)
- Pointing at the inventory to be collected – this is comprised of (for MVS Assets**)
    - **Source language files:**
        - **COBOL, PL/I and Assembler** are supported. Primarily ANSI-standard (aka IBM) dialects
        - It should be noted that RAA will find and document languages such as: CA-Easytrieve, XML, etc. However, these assets are only catalogued in the Repository. No analytics are performed on them.
    - **Copybooks and includes,** plus the proper copybook (SYSLIB) path (called "concatenation sets")
    - **JCL – both runstream and Procs**
    - **Transaction (RDO or IMS GEN) formatted table entries** – these are derived from an ISPF panel (part of the RAA installation process) that reads the system tables and creates an extract file
    - **DB2 table, view and stored procedures** - pulled from the catalog by running an ISPF panel-driven utility

…and scanning/parsing and analyzing the inventory.

Inventory scans produce entries in the RAA repository (a set of ~200 DB2 tables) that contain application "meta-data". Meta-data includes: Elements and Relationships among elements about your application derived from the code that you've fed into the inventory process – and which constitute a static model of your application available for analysis. Errors may (well) occur during inventory collection for things such as missing source, unknown elements, etc. These will most likely need to be resolved – and typically will be by an RAA Admin team that is dedicated to loading and managing the server.

There are – of course – additional steps and procedures necessary to build out a true production-scalable system, including optimizing/tuning the database, setting up security/access to the repository, creating "source-synch" processes for incremental inventory collection tied to application version upgrades, integration with your shop's source management system, etc. For information on what we recommend for these aspects of real-life/large-scale RAA utilization contact us: inquires@royalcyber.com

Analyzing Online Transactions

Once you've gotten the meta-data collected in the repository you can begin doing online transaction analysis. We'll talk through the use of RAA for CICS application/transaction work but (almost) the same sets of features exist for IMS TM applications. The one major difference being that RAA does not capture any meta-data on MFS screens – while it does understand and relate BMS Maps/Mapsets back to Transactions.

Questions you can answer – and assets you can trace from the starting point of an Online Transaction include what are shown in Figure 7. Essentially you can begin by searching for a given Transaction and from there learn/examine the model of the Transaction's elements and dependencies.

** Recall that RAA also supports distributed technologies such as: Java/J2EE and Web Services

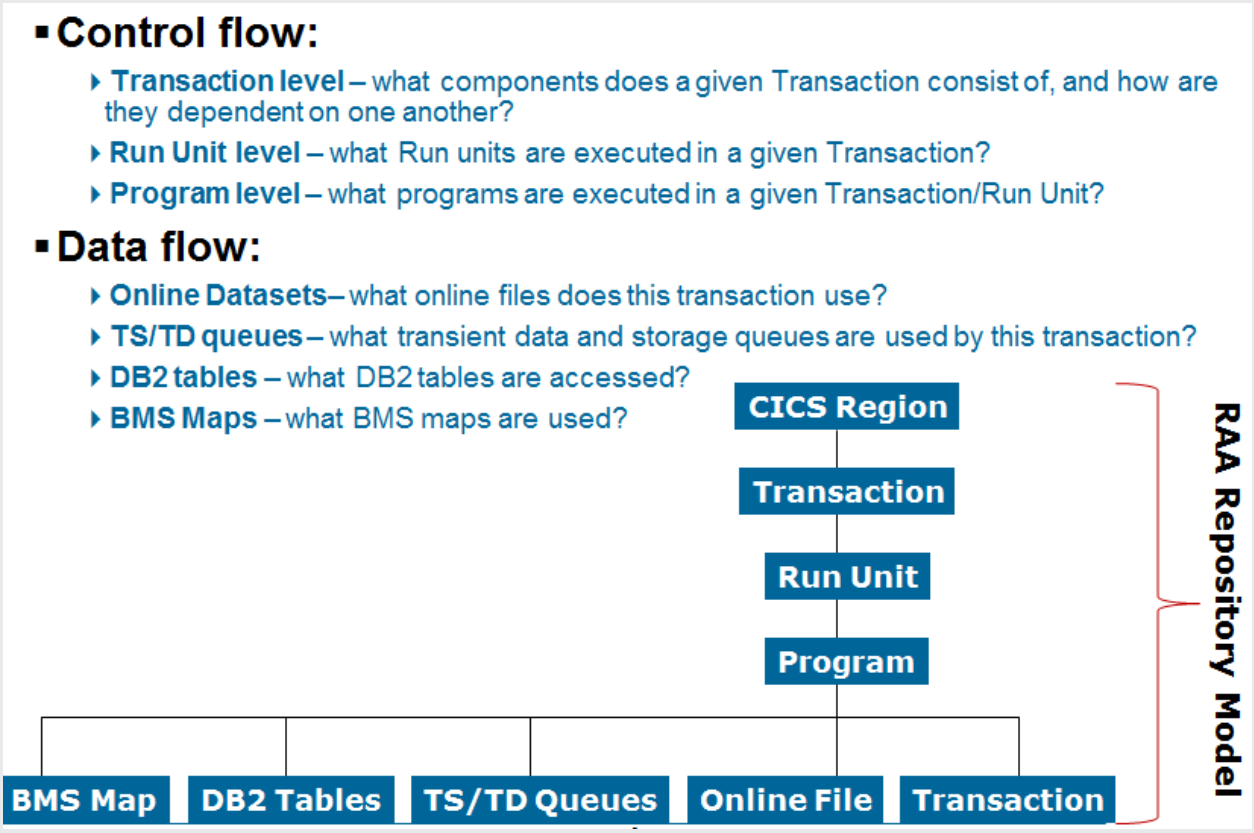


Figure 7 – Transaction Semantics Captured by RAA and rendered through views of your static code stored in the RAA Repository

So you can see from Figure 7 that:

- Transactions are defined in (they execute in) a given **CICS Region**
- A **Transaction** is comprised of one-to-many **Run Units**.
- A **Run Unit** is comprised of one-to-many programs
- Each program can interact with:
    - **BMS Map/Mapsets**
    - **TD or TS Queues**
    - **a VSAM files**
    - **Databases** (DB2 tables/views or IMS segments)
- And a program can (in turn) launch another Transaction.

Not shown in **Figure 7** is that you can also capture what are called "**User Defined Relationships**" – which are manually-specified relationships between any element in the repository and any other element. Manual specification is distinguished from the automatic inventory collection that is the primary means of capturing meta-data.

User Defined Relationships (UDRs) are often utilized for defining relationships between (say) an EJB-Jar file with a CICS transaction. Once defined in the repository RAA automatically associates the UDR with both the EJB and CICS transaction detail pages. UDRs also participate in RAA's Impact Analysis – allowing you to uncover say, what impact a change to a Jar file would have on any CICS transactions that service the EJB.

Steps…

Starting from the RAA home or ***landing*** page, the steps to access CICS Transaction information are:

- From **Explore > MVS assets > CICS online region > Total**
  - Select a **CICS site**
- From the **Transactions** tab use **Search** to identify one or more Transactions you are interested in exploring
- Select (click) a Transaction and from **CICS transaction details > Actions** select: **Show the CICS transaction diagram**, and from the diagram:

Transaction diagrams allow standard GUI mouse actions:

- Hover (mouse) over the various elements in the diagram – to understand what they represent
- Discover the relationships among elements:
- Double-click any of the following, and trace the related assets – studying as much as you are interested of:
  - **Run Units –** *an RAA Run Unit is comparable to a composite load module*
  - **Program**
  - **Online files and/or databases**
  - **BMS**
  - **TD (Transient Data) or TS (Temporary Storage) Queues**

Let's take a look at an example.

An RAA CICS Transaction Diagram

Figure 8 shows a picture of a single transaction (AC01) – and all of the Run Units potentially invoked, BMS Maps/Mapsets and TD/TS Queues that are used in a Run Unit.

To understand this diagram:

- The Run Units are green: ACCT01, ACCT02, ACCT03
- Transactions are beige: AC01, AC02, AC03
- Online files are also beige (but have a different icon): ACCTFIL and ACCTIX
- BMS code is shown in red: ACCTMNU, ACCTMSG, ACCTDTL
- TS Queues are mustard yellow colored: ACERLOG, AC0, ACCTLOG

This **Transaction** diagram looks – on the surface – like quite a complex little drawing but three things about how this?

1. You can collapse individual Run Units down so that the complexity is – to some degree minimized. We've expanded a few Run Units in the Transaction diagram shown in **Figure 8**, in order to give you a better picture of what RAA is capable of

2. I guarantee that most – if not all of your production transactions are factorially more complicated than this example – which ships as sample/learning application code with RAA

3. Consider the amount of time and effort it would take to piece together the semantics shown here: AC01 invokes ACCT01 - which in turn invokes; ACCT, AC02 and AC03. Various programs in the ACCT01 Run Unit use BMS maps: ACCTTL, ACCTERR and ACCTMNU. They also write to: TS queue AC0 (you can tell that it's a write operation because of arrow direction). In order to understand the semantics of even a simple transaction like this:
    - You would be reading and cross-referencing dozens of files, some of them in RDO table s you might have access to
    - You'd be visiting with Subject Matter Experts – expending their cycles
    - And at the end of hours-to-potentially-days of) work? You'd end up with a mental picture that at best matched Figure 8 – which was obtained via a single mouse-click.

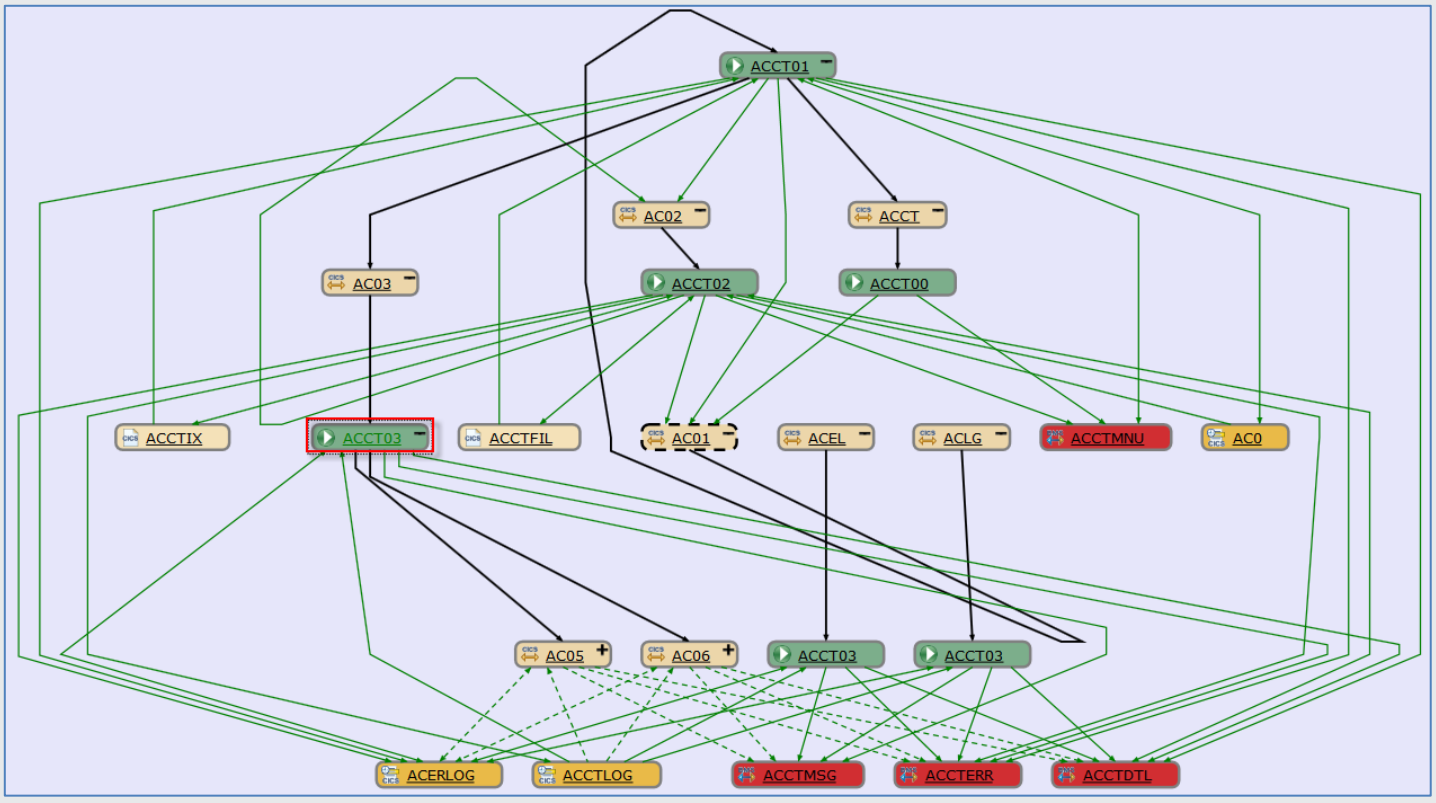


Figure 8 – The partially-expanded semantics of a CICS online application beginning from trancode: AC01

Drilling to analyze the details

As helpful as Figure 8 obviously is – if all of RAA is an apple, its Transaction diagram is one small piece of the skin. By single-clicking the **ACCT03** Run Unit (highlighted in Figure 8) you drill-down into the details of the Run Unit. The first screen you would see is the **Run Unit Details** page (not shown). From an **Actions combo-box** on the Details page, you could generate a **Run Unit diagram** (**Figure 9**).

The Run Unit diagram takes your analysis to another level of precision by exposing - in this case - that 5 CICS transactions invoke the **ACCT03** Run Unit. And that the entry-point into the Run Unit is the program **ACCT03** - shown in blue and highlighted. Other details include **ACCT03** – the program sends 2 BMS maps, writes to a TS Queue and calls **ACCT04**, etc.

If you then clicked **ACCT03** (the highlighted program in the diagram) you would end up at a **Program Details page** – which provides an extensive amount of meta-data on the program (copybooks used, Calling/Called programs accessed, DB2 tables/columns referenced, Batch jobs and Online transactions in which the program participates, etc. There are also 25 Actions including one which describes the logic in the program: a **Control Flow diagram** (Figure 10). This diagram (like the same function in Rational Developer for System z) shows the structure of the program's PROCEDURE DIVISION – and (in the case of **ACCT03**) exposes little execution miracles such as paragraphs **AC05** and **AC06** GO TO'ng their paragraph labels recursively (note the boxy arrows that flow from the back of AC06/AC05 to the front of the paragraph). As an aside, each paragraph is hyper-linked (thus the underlined paragraph name). If you click a paragraph from a Control Flow diagram the program source opens – and the paragraph highlighted.

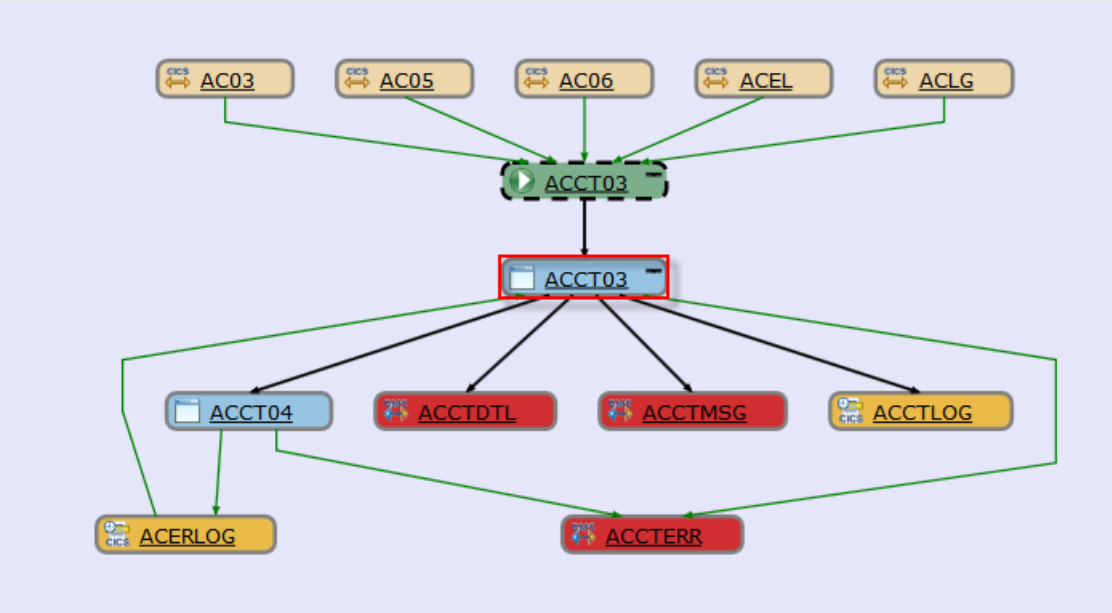


Figure 9 – An RAA Run Unit (ACCT03) – showing specific relationships and dependencies

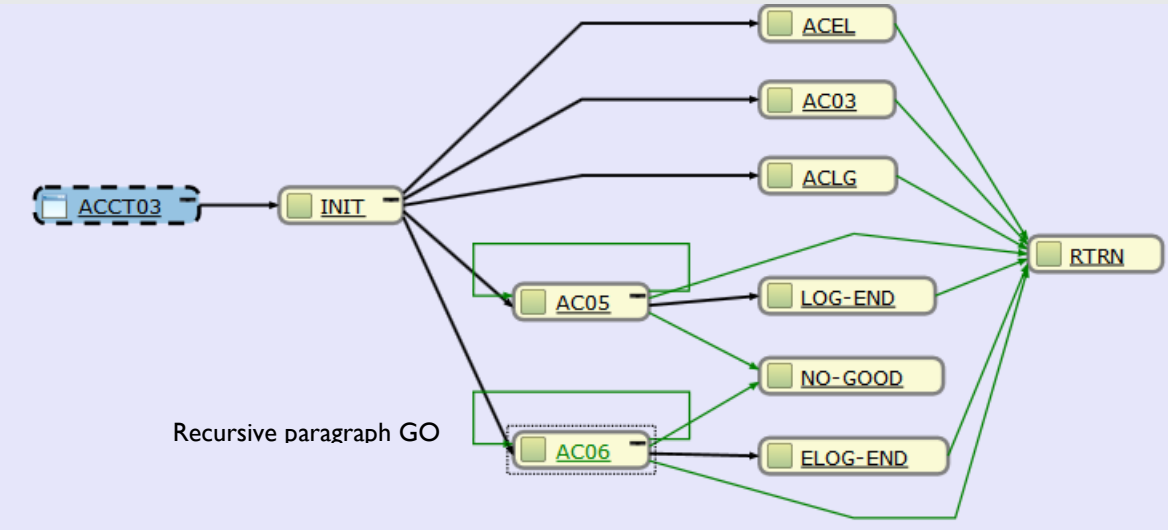


Figure 10 – An RAA Program Control Flow diagram of the ACCT03 program

Our view

So, utilizing the technique of drilling down from higher to lower levels allows you to grasp the semantics of: Transactions ➔ Run Units ➔ Program logic at a glance … to understand the dependencies and relationships among various application elements and to hyperlink to any point of interest in the repository from a single mouse click.

It's funny… as I proofread that sentence after I wrote it, the phrase "… ***from a single mouse click***." seemed a little "Madison Avenue", y' know… just a little too… *effusive*. But honestly? The more we use RAA in our work (and we do… use RAA on Royal Cyber z/OS engagements) the more it seems like RAA is some sort of IBM "**best kept secret**". And in describing RAA's value to z/OS maintenance and support programmers doing common/every day project work – and in representing what it does with phrases that are a bit over the top … well … it doesn't exactly seem like a crime.

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

However, 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.

Click to contact us about our RAA service offerings and request an online/remote demo. After the demo judge for yourself if this article is "effusive" – or maybe a little understated in its view of RAA's value to z/OS traditional maintenance, production support and development work.

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 specialist skills in writing Eclipse plug-ins to RDz.

Royal Cyber in the News

In February we delivered our first IBM RDz Distance Learning classes – to over 150 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 March Distance Learning session, but you can sign up for April (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
- To view our technical newsletter - Volume 1, January 2014 – including additional deep-dive articles on RDz and RAAi: http://royalcyber.org/mn/index.html

What people are saying about Royal Cyber's RDz instruction:

The instructor was awesome! He obviously knew this tool inside and out and was talented at teaching others about it. His relaxed and approachable attitude encouraged much discussion and questions….which is always a huge benefit in any class in my opinion.

I found it to be an excellent class… especially in the two sessions per day environment, which allowed me to apply the session in my own shops environment and retain the practical knowledge.

The class worked out great; I really appreciated the format. The instructor was very knowledgeable and did a good job with the challenging task of lecturing via phone.

The Royal Cyber online assistants were great because they could keep a running chat going, answering questions without interrupting the instructor's talk.