QuickRDA Modeling System

Diagram & Report Generation Guide

Erik Eidt

Version 2.7

March, 2011

***Copyright (c) 2011, 2012 Hewlett-Packard Development Company, L.P.***

[Diagram & Report Generation Guide 5](#_Toc283061708)

[Sourcing Methods 6](#_Toc283061709)

[Combining Graphs from multiple workbooks 7](#_Toc283061710)

[the Build table 8](#_Toc283061711)

[Include Style 9](#_Toc283061712)

[Visible Inclusion 9](#_Toc283061713)

[Invisible Inclusion 9](#_Toc283061714)

[Cursor Position on the Build Table 10](#_Toc283061715)

[Build Switches/Options 11](#_Toc283061716)

[Abstraction 11](#_Toc283061717)

[Filtering by Layer 11](#_Toc283061718)

[Plan Inferencing 12](#_Toc283061719)

[Diagram Debugging 12](#_Toc283061720)

[Orientation 12](#_Toc283061721)

[Reporting 12](#_Toc283061722)

[Advanced Diagram Abstraction 13](#_Toc283061723)

[Diagram-Abstraction Filters 14](#_Toc283061724)

[Overview of the Directives 17](#_Toc283061725)

[Select Directives 18](#_Toc283061726)

[Union Directives 18](#_Toc283061727)

[Avoid Directives 18](#_Toc283061728)

[Reach Directives 19](#_Toc283061729)

[Reach1 Directives 19](#_Toc283061730)

[Cull Directives 20](#_Toc283061731)

[Graphing – the RDA Start Browser Button 21](#_Toc283061732)

[The Link Back Feature 21](#_Toc283061733)

[The Highlight Feature 21](#_Toc283061734)

[Diagram Debugging 22](#_Toc283061735)

[References 23](#_Toc283061736)

***Table of Tables***

[Table 1. BNF for Diagram-Abstraction Filters 15](#_Toc283061737)

***Table of Figures***

[Figure 1. The Build Table Worksheet 8](#_Toc283061738)

# Diagram & Report Generation Guide

This section is fairly incomplete. When it’s done, it is intended to describe the rather mechanical usage of the tool, leaving the definition RDA contextual architecture, as well as the struggles of elicitation and getting to good granularity and correctness for another paper.

## Sourcing Methods

The QuickRDA tool will function in one of two modes for choosing source graphing:

* Build Table —a kind of makefile, is the preferred method
* All open Excel Workbooks & Worksheets, deprecated

If it finds a build table in the current excel workbook (from which the button has been pushed) then it will follow that as guidance on what to graph. Failing finding a build table in the current workbook, it will graph every worksheet of every currently open workbook.

The all open workbooks method is an older method that precludes the use of most of the options in the build table, and is therefore deprecated.

*.*

## Combining Graphs from multiple workbooks

Simply source multiple filled-in workbook forms in the same copy of the excel application, and the graphing options will merge graphs. Note that by default with an Excel installation, all spreadsheets are loaded into the same instance of the excel application. This makes it easy to combine QuickRDA templates — simply use the graph while they are all open.

*Newer templates come with a pre-defined Build sheet.*

*Because by default, all workbooks that are currently open share a single copy of Excel, it is best (at least for now) to close all unrelated Excel spreadsheets and workbooks.*

## the Build table

The build table works like a makefile, specifying build targets from sources. The smallest unit of source is a single worksheet (tab).

The build table identifies how sources (worksheets) are combined into output graphs. It allows multiple sources to be combined into a permutation of outputs. One view (output) can have some filtering, another not. One view may show from only one source, another may combine several.

The Build table is also an Excel table. It must have the Excel table name QRGraphSpec. Current templates come with such a build table in a Build worksheet.

The build table consists of rows each of which is used to specify either a sources or a build option. Sources are specified by the Workbook and Worksheet columns. A blank workbook means the current workbook. A blank worksheet means all worksheets in the workbook. The other columns, start with the special name M1 (the actual column header names beyond M1 are irrelevant). Each column, beyond and including the M1 column, specifies a model (diagram or view) that is generated. Cell values indicate whether to include source (with an x), or an option name, which starts with a / character.

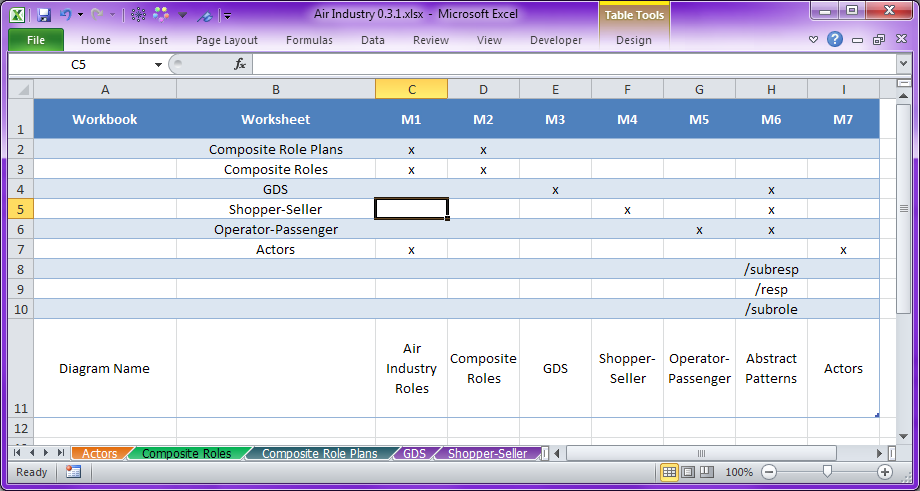


Figure 1. The Build Table Worksheet

### Include Style

Using a check mark in a cell, sources, tabs or worksheets, and workbooks, are added to the diagram. There are two styles of include: visible and invisible. Visible inclusion is the normal mode. However, for a number of purposes, we may want to include but not graph certain items. This brings the information in the tab into view of the reasoning engine in the tool, but does not show the included items in the graph.

#### Visible Inclusion

Normally, we choose sources to bring into the graph and show. Using an “x” in the cell selects this normal visible inclusion mode.

#### Invisible Inclusion

To include sources in the graph exposing them to the tool’s reasoning engine as well as making them available for Advanced Diagram Abstraction, use an “i” character in the cell. These sources are collected as information but not displayed by default. The reasoning engine and other build switches can be used to draw on the “invisibly” included information.

### Cursor Position on the Build Table

Making cell selection within the Build table of a column for a build target instructs the build button to graph only that one build target. Moving the current selection for the Build worksheet out of the build columns (to empty space at the right, or before the M1 column to the left, e.g. cell A1) instructs the system to build all targets. Note that this selection continues to apply even after leaving the Build worksheet for another worksheet in the workbook, to enable repetitive graphing of the same build target.

### Build Switches/Options

#### Abstraction

There are currently three contextual layer filtering options, which abstract information in order to present overview versions. The filters are often combined in the order listed here.

* /subresp — promotes providing and consuming of artifacts to the top-level responsibility in its chain, while also hiding these non-top-level responsibilities.
* /resp — promotes providing and consuming of artifacts by top-level responsibility to its assigned role, while also hiding these top-level responsibilities.
* /subrole — promotes providing and consuming of artifacts by roles to the top-level role in its chain; this is particularly useful to observe the high-level picture after a role has been decomposed by providing zoom detail.

#### Filtering by Layer

* /restrict=BCA — restricts the diagram to show only elements derived from the namespace supplied as XXX; this is particularly useful when a model contains multiple layers, and the author chooses to adorn a source having mostly Contextual description with mappings to Conceptual (or vice-versa), rather than repeating in another source file the portion of the Contextual (or Conceptual) description that would be necessary to establish the mapping. Meaningful namespaces are:
  + BCA — Business Context Architecture; only elements of the RDA Contextual Layer will be shown
  + CSA — Conceptual Service Architecture; only elements of the RDA Conceptual Layer will be shown

#### Plan Inferencing

* /ipoff — turns off generation of roles in plans (on by default).

#### Diagram Debugging

* /tdoff — turns off automatic omission of differentiators (on by default).
* /hideoff — turns off all automatic hiding, such as for inferencing for plans, running abstraction filters, removing superfluous assigned-to’s in responsibility decomposition, or hiding containment linkages (as normally done when showing containment).

#### Orientation

* /lr — change graph orientation from default top-to-bottom to left-to-right
* /compup — change Component Of placement from default down to up
* /nocont — skip containment inferencing (on by default), which has the effect of showing containers as separate nodes with linkages to containees instead of containment as enclosing containees (usually with the linkages suppressed).
* /atob —make “Owned By” have be an attaching property instead of grouping property (default)

#### Reporting

Reports work best when including all of the sources, even if the resulting diagram is complex. It is recommended to create a separate build table column for report generation that includes all the sources (with the exception of a custom styles tab).

* /report — generates a new spreadsheet that evaluates the model. Best use is to include all sources in report generation: let the evaluator see all the separate sources as a single whole.

## Advanced Diagram Abstraction

Diagramming abstraction is the ability, from the same given set of sources for a model, to create one or more diagrams, each showing only custom portions of particular interest.

For example, some number of roles, responsibilities, and artifacts may be described in a single source (tab or worksheet). Sometimes we’d like to create a diagram focusing on only one role and its responsibilities from that source. Other times, we’d like to focus just on the interaction between two roles in that single source.

And because the complete picture of a single role or the interactions between two roles may span several sources, we’ll want to be able to combine one source (tab or worksheet) with others, and then do the same: focus on a single role or, perhaps on the interconnection between two roles.

Such diagramming abstraction is supported in QuickRDA through a mechanism called Diagram-Abstraction Filters.

### Diagram-Abstraction Filters

Diagram-Abstraction Filters are QuickRDA’s mechanism to create a separate diagram whose focus is on specific portions of interest in a model by revealing that interest while hiding the rest. Multiple diagram-abstraction filters can be applied within a single diagram. A single filter itself is composed of clauses that combine to reveal the specified portions of interest. Each clause has a directive and an optional argument list, which is used to specify zero or more given targets.

An analogy can be made with SQL’s WHERE clause: diagram-abstraction filters are written to identify the focus of interest. The analogy with SQL further holds with SQL’s FROM clause: a set of initial sources must be specified over which the filters can take place. This is done by specifying the sources of interest to filter over using the build table; the diagram-abstraction filters themselves are also specified in the Build table.

As already mentioned the diagram-abstraction filters are used to identify the desired focus of interest to reveal — rather than identifying portions of disinterest to hide. Because of this, when using diagram-abstraction filters, *at least one — and commonly all — the chosen sources to filter over should be included with the “i” option (*Invisible Inclusion*) instead of the “x” option (*Visible Inclusion*).* (Of course, one can mix usage of the “i” option for some sources with the “x” option for others.)

Though the analogy with diagram-abstraction filters can be made with SQL’s where clause, the way these filters compose is slightly different, as they are designed to work with complex tree structures and tangled webs that are the nature of models or graphs, rather than the more regular tabular structure found in SQL. To facilitate this, multiple diagram-abstraction filters can be composed, with each filter increasing the revealed portion of the model.

Further, each individual diagram-abstraction filter is itself composed of any number of clauses, which contain a directive and specified given targets (parameters). Each clause in the filter composes to influences the next as follows: each clause identifies a set of graph elements, to which the next clause applies, known as the application set. The clauses each contribute in some way to the overall set that is revealed by their filter.

The first clause of a filter applies to the accumulated revealed set by preceding filters. The identified set of the last clause is discarded, with the result of one whole filter being a revealed set. The diagram accumulates all the filters taken together.

The first clause of any filter receives the whole revealed graph as its input set — that is, the graph as revealed by preceding filters (which may be nothing) together with the graph as revealed by the visible inclusions in the build (which also may be nothing).

Advanced filters are written as a series of clauses that apply to each other. A filter takes the form of “/” character followed by one or more semi-colon separated clauses. A clause takes the form of a directive with a target list. In BNF the syntax is:

|  |
| --- |
| Advanced-Filter ::= “/” Clause-List  Clause-List ::= Clause | Clause “;” Clause-List  Clause ::= Directive “=” Target-List  Target-List ::= Target | Target “,” Target-List  Target ::= [ Qualifier-Name “:” ] Name-Or-Type-Value  Qualifier-Name ::= Identifier  Name-Or-Type-Value ::= Identifier | “-” Identifier | Identifier “-”  Directive ::= “SelectNode” | “SelectType” | “SelectLink” |  “UnionNode” | “UnionType” | “UnionLink” |  “AvoidNode” | “AvoidType” | “AvoidLink” |  “ReachNode” | “ReachType” | “ReachLink” |  “Reach1Node” | “Reach1Type” | “Reach1Link” |  “Reach1Forward” | “Reach1Back” |  “CullNode” | “CullType” | “CullNode” |

Table 1. BNF for Diagram-Abstraction Filters

QuickRDA will apply the directive of the clause to the selected target, during which, any nodes identified are taken as applying to subsequent clauses.

A filter to reveal a role node, Travel Operator with all its responsibilities and their artifacts is constructed as follows:

/SelectNode=Role:Travel Passenger;Reach1Link:Assigned To;Reach1Type:Artifact

Artifacts reachable from whatever is assigned to the specified given role(s) will be revealed (along with the responsibilities and their role assignments).

The following two filters each reveal a different portion of the model for the diagram:

/SelectNode=Role:Travel Operator;Reach1Type=Responsibility;reach1Link=Consumes

/SelectNode=Role:Travel Passenger;Reach1Type=Responsibility;reach1Link=Provides

Only Consumed Artifacts reachable from Responsibilities of Travel Operator will be shown, along with Provided Artifacts reachable from Responsibilities of Travel Passenger will be shown.

A command to reveal the connection between two roles is:

/SelectNode=Role:Travel Operator;ReachNode=Role:Travel Passenger

This will reveal any and all paths in the graph that connect the two roles (regardless of the direction of the arrowheads.)

Limiting the connections between two Roles to only non-role entities can be very useful:

/SelectNode=Role:Travel Operator;AvoidType=Role;ReachNode=Role:Travel Passenger

To show all responsibilities (and artifacts associated) of one role, along with its connection to another role:

/SelectNode=Role:Travel Operator;Reach1Type=Responsibility;Reach1Type:Artiact

/SelectNode=Role:Travel Operator;ReachNode=Role:Travel Passenger

#### Overview of the Directives

Directives can be thought of as the keyword phrase that introduces a clause of a diagram-abstraction filter. Clauses are composed of directives and generally, though not always, specified with one or more given targets as parameters. A directive tells the clause how to interpret its parameters what set to identify and reveal given the set it applies to (from previous clauses) along with the rest of the graph.

Directives usually come it sets of three forms to choose from: node forms, type forms, and link forms. The node forms specify the names of nodes; the type forms specify types of nodes. The link forms specify properties.

For link forms, properties can be specified as:

* plain — in which case nodes that form both or either the subject and object of, e.g. on either side of a link, by that property are identified. This form is specified using no minus signs, e.g.

Name-Or-Type-Value ::= Identifier

* prefix — in which case nodes that are the subject of the property are specified. This form uses a minus sign preceding the property identifier.

Name-Or-Type-Value ::= “-” Identifier

* postfix — in which case nodes that are the object of the property are specified. This form uses a minus sign succeeding the property identifier.

Name-Or-Type-Value ::= Identifier “-”

When using link forms to restrict node identification to property subjects (prefix form) or property objects (postfix forms), it is useful to remember the nature of informational relations or links, captured and stored by the QuickRDA tool.

Every relation (or link), connects a subject with an object via a property; these connections are directed, in the sense that one of the nodes linked together is the subject and one is the object, and the subject and object are not interchangeable. Generally speaking, the QuickRDA tool will diagram the arrowheads pointing to the object; thus, you can read the information of a relation or link as subject (having the arrow tail), link (the property), object (with the arrowhead) — as if it was a simple infix statement in natural language.

However, for most purposes, the direction of the arrowhead only indicates how the text of the label should be read — more specifically, the arrowhead direction does not indicate ownership or flow of control. Thus, it would be inappropriate for the directives of diagram-abstraction filters as a whole to regard the direction of the arrow in identifying and revealing nodes in the graph — unless specifically instructed to do so.

#### Select Directives

The select directives directly reveal elements or portions of the graph. They ignore their application set, identifying and revealing only the specified given items. Because of this, select directives are often used in the first clause of a diagram-abstraction filter. The items they identify apply to subsequent clauses in the same diagram-abstraction filter.

SelectNode — this directive identifies and reveals the specified given nodes.

SelectType — this directive identifies and reveals the nodes of the specified given type.

SelectLink — this directive identifies and reveals the nodes on either side of the specified properties.

#### Union Directives

Union directives differ from their respective select counterparts in that they do not ignore their input set, instead accumulating the current input set together with a specified given items that are identified.

UnionNode — this directive identifies and reveals the specified given nodes.

UnionType — this directive identifies and reveals the nodes of the specified given type.

UnionLink — this directive identifies and reveals the nodes on either side of the specified properties.

#### Avoid Directives

The Avoid Directives reveal any nodes: rather they inform and modify the behavior of the next Reach (or Reach1) Directive. Further, they pass their application set through unmodified. One or two or all three of the Avoid Directives forms (Node, Type, Link) can be specified and will take cumulative effect on the next Reach (or Reach1) Directive. Avoid modifiers expire after the next Reach (or Reach1) Directive (they also do not carry over from one diagram abstraction filter to another).

AvoidNode — this directive informs the next Reach or Reach1 directives not to consider paths through the specified given nodes.

AvoidType — this directive informs the next Reach or Reach1 directive not to consider paths through nodes of the specified given type.

AvoidLink — this directive informs the next Reach or Reach1 directive not to consider paths through the nodes on either side of the specified properties.

#### Reach Directives

These powerful directives are used to diagram the path (via nodes & edges) between any specified given items. They identify and reveal every path reachable between the sources (the set they apply to, taken from the previous clause) and the parameters (the specified given targets). *Because they reveal not just one path, but every connecting path, often they need to be restrained: this can be done with Avoid Directives, which modify the behavior of the Reach Directives.*

ReachNode — this directive identifies and reveals all paths between the input set and the specified given nodes, e.g.

ReachNode=Role:Travel Passenger

ReachType — this directive identifies and reveals all paths between the input set and nodes of the specified types. *(Note: ReachType is not implemented yet.)*

ReachLink — this directive identifies and reveals all paths between the input set and nodes on either side of the specified properties. *(Note: ReachLink is not implemented yet.)*

*Note also that all paths between each of the original sources and the final targets are identified and revealed; however paths between the original sources themselves, and the original targets themselves, are not specifically revealed (if desired that would require additional filters or clauses).*

#### Reach1 Directives

These are a limited version of reach directives: they reach only one node away from the current set they apply to.

Reach1Node — this directive identifies and reveals the nodes that are one link of distance away from their application set. Unlike most other xxxNode directives, this one does not use parameters. Therefore, the = … specification is not used.

Reach1Type — this directive identifies and reveals nodes of the specified given types that are one link away.

Reach1Link — this directive identifies and reveals nodes that are one link away using only the specified given links.

#### Cull Directives

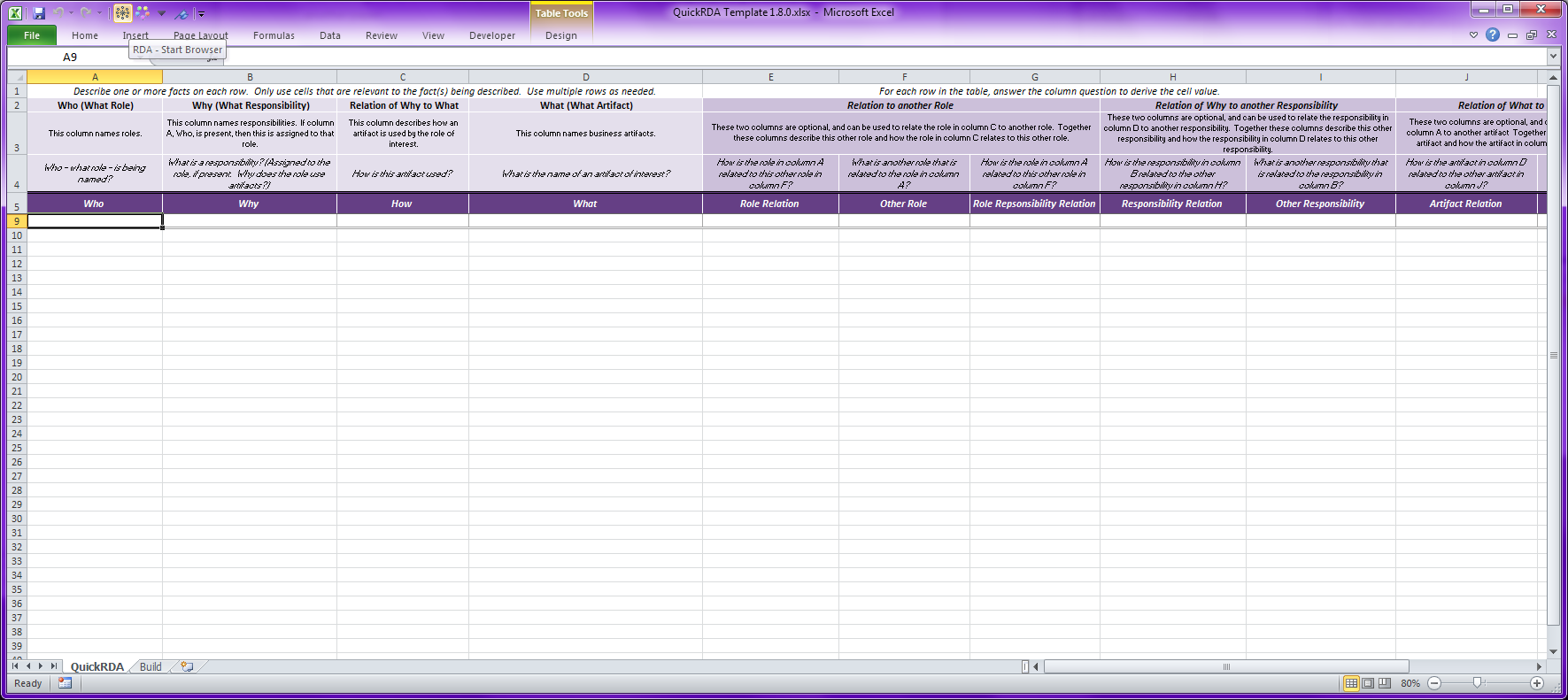
Cull Directives have no effect on the current graph: instead, they pass through their input set modified by removing items of the specified given kinds as their output set.

CullNode — this directive removes the specified given nodes in passing its input set on as its output set.

CullType — this directive removes nodes of the specified given type in passing its input set on as its output set.

CullLink — this directive removes nodes connected by of the specified given link in passing its input set on as its output set.

# Graphing – the RDA Start Browser Button



The first button generates an .svg file, and launches a viewer for it.

Though it is not strictly necessary — as the generation takes place within the working copy of excel — remember to save your work occasionally (or more frequently ;).

## The Link Back Feature

There is a new feature supported by some viewers that assists in round-tripping data entry. The first graphing button generates the .svg file and launches a viewer. If the viewer is capable of following hyperlinks, you can click on any node or edge, and the viewer will return you to the row in the spreadsheet that (first) declared that concept or relationship.

The link back feature works only if the necessary workbooks are already open. It also only works for the most recently drawn graph. Thus, the intended usage is after graphing a single diagram and while the workbooks are still open.

## The Highlight Feature

The highlight feature helps completes round-tripping between the diagram and the spreadsheet source. It highlights in the graph any facts defined in the rows involved in the current selected in Excel at the time that the build (Start Browser) button is pressed. These facts appear in the diagram with a fuchsia color whose lines are wider than they otherwise would be so they really stand out even when the diagram shrunk so that more of it fits on the screen.

# Diagram Debugging

For debugging purposes several of the build table options are useful.

One is /tdoff, which turns off trimming of differentiators. This is particularly helpful, of course, when using differentiators, as sometimes we intentionally graph separate nodes with the same name though when things are in error it’s hard to determine which is which.

The other is /hideoff, which turns off hiding information that was caused by inferencing. This is particularly helpful, for example, in the construction of conceptual architecture, as a service that is said to be in multiple towers (which aren’t sensible) will be graphed outside of either tower; similar is true for the same operation said to be in multiple services. Using this option allows seeing the multiple linkages that caused it to be graphed this way.

# References