

Amendment 01, Questions and Answers:

Question: Our company would like the opportunity to respond to Solicitation W9124Q-OLAP. Is there a more detailed SOW available?

Answer: No. The requirement is being solicited using the multi-step procedures found at Federal Acquisition Regulation 15.212-2. This notice shall be considered Phase I of the selection and award process. If an offeror thinks that he may have an OLAP system that can provide us with improved capabilities, we ask that each interested source submit the evidence as noted in the synopsis.

Question: (The synopsis) stated that there will be 50 concurrent users. What is the total number of users?

Answer: Total figure is also 50.

Question: Terminology like 'threat shots', 'threat systems', 'blue systems', 'blue infantry', 'blue personnel', 'blue dismounted infantry', 'blue joint systems', 'blue FCS systems', 'blue manned ground vehicles', and 'high priority targets' assume a certain level of knowledge of the data that is not clear simply by looking at attribute names.

Answer: In the ES file the field FSIDE is the side of the firer and TSIDE is the side of the target. Blue forces show side 1 and Threat forces show side 2. In some cases, such as indirect fire, there is no actual target. In this case, there will not be a TSIDE value.

A shot is determined when the ES field RPB has a value greater than 0. A hit is determined when the ES field HM has any one of the following criteria: 'HIT-B', 'HitF', 'HIT-H', 'HitK', 'HitM', 'HitMF', 'HitNoK', 'HIT-T', 'L-HIT', 'WAM'. A kill is determined when the ES field KILLS has any one of the following criteria: 'F', 'K', 'M', 'MF'. An acquisition is determined when the ES field ACQ has any one of the following criteria: '0', 'A', 'D', 'I', 'R', 'RL'. The meanings for the different levels are unimportant.

The categories come from the attached CAS table. In the CAS table the field SYSTEM_TYPE tells what the criteria is, i.e. Infantry, Armor, Manned Vehicles, HPT, etc. There can be multiple CAS tables, because if we define a unit more than once in any one table, then the query often double counts results. Meaning, if a system is defined as Armor and Non-Infantry it will show up once in the Armor and once in the Non-Infantry columns. If the given CAS table does not have the names referred to in the MoEs then I can provide more CAS tables or disregard the MoE.

Personnel refers to

Systems are broken out by their unit ID or bumper number. B12345 is in the 1st Brigade, 2nd Battalion, 3rd Company, 4th Platoon, and 5th Squad. Consider FCS and UA comp systems from Brigade 1. Consider UE from Brigades 5-8. Consider Joint from Brigade 9. CABs are defined as Combined Arms Battalions. They are from the 1st Brigade. 1st CAB is from the battalion 1, 2nd CAB is from battalion 2, and 3rd CAM is from battalion 3.

Question: The basic difference between an acquisitions and engagement is not clear.

Answer: In the ES file the field DATATYPE differentiates between engagements and acquisitions. 1 refers to an acquisition and 2 refers to an engagement. A complete listing of field headers, with descriptions is provided in the PowerPoint package. Also, an acquisition will include data in the "Sensor" column and an engagement will include data in the "Munition" column.

Question: You define unique acquisitions as the first time it was acquired but again it is not clear exactly what that means from a data perspective.

Answer: A Unique Acquisition is the first instance a system ID number (TUNIT, for the ES Table) is detected. Every entity in the battle is assigned a unique ID number. We know the record is an acquisition event because DATATYPE = 1.

We use the field TIME to determine the first acquisition, as the field TIME shows the model time when events occur. To determine the time an entity is first acquired, we look for the minimum time for that ID number (TUNIT) where the DATATYPE = 1.

Question: References to aspect, ranges, and time intervals would seem to infer the creation of interval groups based on the different data elements. With the exception of 'aspect' it is not clear which data attribute the interval group is based on (e.g. Time?).

Answer: Intervals are derived from the ES file fields, ASPECT, FIRRNG, and TIME. Aspect angle is from the ES field ASPECT, range is from the ES field FIRRNG, and time is from the ES field TIME. A complete listing of field headers, with descriptions is provided in the PowerPoint package. The acquisitions, shots, hits, or kills are accumulated based on the interval the user selects.

Question: There is reference to Phase I - III groupings but based on what attribute?

Answer: Phase times are associated with the field TIME. CASTFOREM time is always measured in minutes. Refer to the PowerPoint package for specific phase time criteria.

Question: I think it is necessary to more clearly define what each MoE is looking for, where the data comes from, and the filtering criteria used to isolate exactly what the MoE is asking for.

Answer: The data in question comes from two sources. The ES, MT, US, and UL files are model output files. Currently, the model output flat text files to certain folders of our networks. Our post-processors select the folder and import the text files. Generally, we have to unzip the files to import them into our database. The other files (CAS, CAF, CAM, TA, and Force Structure) are generated by the post-processor and are based off the US and UL files. A user goes through a wizard that allows them to define systems and munitions into categories.

The data output files in question are not like standard transactional database files. The tables are not normalized and there have been no attempts to make them so. There are no primary keys or foreign keys. There has been talk about future simulations outputting files into a relational database. Should this happen, the query management and optimization will improve. However, currently we are using files similar to what was provided.

We have created various files to help us with our analysis. The CAS, CAM, CAF, TA, and force structure tables are created by the user not by the model as the ES and MT files are. The CAS, CAM, CAF, TA, and force structure tables are populated by the user and are based off selected US and UL files. There can be multiple CAS, CAM, CAF, and TA tables. The user should be able to select the desired table (or create new ones) from a list and have dynamic results appear for the selected tables. For example if selecting the CAS-Infantry table our firers and targets would be broken out into categories of INFANTRY and NON-INFANTRY. If the CAS-HPT were selected, the firers and targets would be broken out by High Priority Targets classifications. These tables essentially allow us to format data easier, because the query pulls information from the CAS, CAM, and CAF tables and links it to the ES and MT files.

Example: A query is created to show the kills by certain firers for certain targets. Two instances of the CAS table are created and linked to the ES table. The links are created by the TYPE_UNIT (CAS) – TFUNIT (ES) and TYPE_UNIT (CAS-1) – TTUNIT (ES). Notice we include the TFUNIT (ES) and

SYSTEM_TYPE (CAS) and TTUNIT (ES) and SYSTEM_TYPE (CAS-1) in the output.

The screenshot shows a database query interface. At the top, there are three table lists: 'CAS-ALL', 'ES-Sample-S1', and 'CAS-ALL_1'. 'CAS-ALL' and 'CAS-ALL_1' have fields: TYPE_UNIT, SIDE, SYSTEM_TYPE, HVA, HPT, ID. 'ES-Sample-S1' has fields: DATATYPE, REP, FIRETIME, TIME, FSIDE, FUNIT, TFUNIT, FIREX, FIREY, FIREZ, SENSOR, MUNITION, TSIDE, TUNIT, TTUNIT, TGTX, GTY, TGTZ. Below these is a query criteria table:

Field:	DATATYPE	TFUNIT	SYSTEM_TYPE	TTUNIT	SYSTEM_TYPE	KILLS
Table:	ES-Sample-S1	ES-Sample-S1	CAS-ALL	ES-Sample-S1	CAS-ALL_1	ES-Sample-S1
Sort:						
Show:	<input type="checkbox"/>	<input checked="" type="checkbox"/>				
Criteria:	2	In ("Firer_System_1", "1")		In ("Target_System_1")		In ("K", "F", "M", "MF")
or:						

This is the output from the above query. We can now pivot off the category system type.

TFUNIT	CAS-ALL.SYSTEM_TYPE	TTUNIT	CAS-ALL.1.SYSTEM_TYPE	KILLS
Firer_System_1	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_1	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_2	Firer Category 4	Target_System_2	Target Category 1	K
Firer_System_1	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_1	Firer Category 3	Target_System_3	Target Category 2	K
Firer_System_1	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_1	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_1	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_3	Firer Category 1	Target_System_4	Target Category 3	K
Firer_System_4	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_4	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_4	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_5	Firer Category 5	Target_System_4	Target Category 3	K
Firer_System_5	Firer Category 5	Target_System_4	Target Category 3	K
Firer_System_2	Firer Category 4	Target_System_2	Target Category 1	K
Firer_System_5	Firer Category 5	Target_System_4	Target Category 3	K
Firer_System_4	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_4	Firer Category 3	Target_System_1	Target Category 3	K
Firer_System_3	Firer Category 1	Target_System_4	Target Category 3	K
Firer_System_3	Firer Category 1	Target_System_4	Target Category 3	K
Firer_System_3	Firer Category 1	Target_System_4	Target Category 3	K
Firer_System_5	Firer Category 5	Target_System_4	Target Category 3	K

Each data output file shows a different aspect of the battle, and captures either different information, or the same information in a different way. The ES file shows engagement and acquisition events throughout the battle. The MT file shows this as well. However, the MT file shows additional information the ES does not have, and the ES file shows information the MT file does not have. We should be able to query most MoEs from either the ES or the MT file, but it depends on the question being asked that tells us which file is better suited to extract the answer from.

Attached is the methodology on how we create our queries.

Chapter 5. TRAC HEAT Queries

The **TRAC HEAT** program retrieves data from the EA Summary files using SQL queries. The queries are either created by the user or imbedded in the VBA code. Common types of queries are:

- **Shots** – Calculate the total number of shots fired by a unit type, unit number, or munition against one or more targets.
- **Hits** - Calculate the total number of hits fired by a unit type, unit number, or munition against one or more targets.
- **System Kills** - Calculate the total number of kills by a unit type, unit number, or munition against one or more non-personnel targets.
- **Personnel Kills** - Calculate the total number of kills by a unit type, unit number, or munition against one or more personnel targets. Personnel kills may be against mounted infantry, dismounted infantry, or system crew.
- **Range** – A simple query modification can calculate all of the above over specific ranges.

The MOE GENERATION form creates these queries (except for range) using the template SQL statements defined below and user selections for the specific firer and target codes. The following samples show how to develop the SQL statements and a sample output table. The output tables with sample data demonstrate the layout of the query results.

Acquisition Queries

Acquisition Query SQL Statement Example

Acquisition Query Output Table Example

Shots Queries

In order to count the number of shots, the user needs to define what constitutes a valid shot. It helps to define the criteria for a valid shot in English before translating them into an SQL statement.

Valid shots require all of the following conditions:

1. A munition must exist for a specific engagement.
2. Queries must count a munition firing only once. Some munitions, like grenades and artillery, can hit multiple targets with each shot. The EA Summary shows each target hit on a separate, but the program must not count every line.
3. A munition may fire multiple rounds per burst (RPB). For example, the query must count a munition with an RPB = 5 as five shots, not one.
4. The user must be able to select specific firers, munitions, and targets.
5. The information must be organized and totaled by specific criteria like "totals for each munition against each target" or "totals for each replication".

These five conditions define a valid shot. Now they need to be translated into SQL:

1. Specific conditions for Munitions are defined in the `WHERE` statement. To check for all cases where a Munition exists, use one of these two statements:

```
WHERE MUNITION <> ""  
WHERE MUNITION IS NOT NULL
```
2. The EA Summary file allows the query to check which line specifically fired the shots and which lines showed multiple hits from that shot. Multiple hits with the same shot

will have the RPB = 0. Therefore, the query must only count the occurrences where the RPB > 0.

```
WHERE RPB > 0
```

3. Since multiple RPB must be totaled, the query must SUM the RPB, rather than simply COUNT all of the occurrences. Calculation statements such as SUM, COUNT, MIN, MAX, AVG, and STDEV are used in the SELECT statement. The "as SHOTS" portion gives the summation column a heading called "SHOTS".

```
SELECT SUM(RPB) as SHOTS
```

4. The Table 5-1 below shows the EA Summary fields for each firer, munition, and target.

System	EA Summary Field
Firer Type (i.e. Tank)	TFUNIT
Individual Firer (i.e. That tank over there)	FUNIT
Round fired	MUNITION
Target Type	TTUNIT
Individual Target	TUNIT

Requesting information on specific units is done in the WHERE statement. SQL understands Boolean logic. If a condition is true, the SQL statement selects that record. Parenthesis in combination with AND, OR, >, <, =, IS, NOT can be used to define complex conditions.

```
WHERE (TTUNIT IS NULL OR (TTUNIT IN (List of Targets))) AND  
(MUNITION IS NOT NULL AND MUNITION IN (List of Munitions))
```

Why is the WHERE statement looking for null TTUNITs? Can **CASTFOREM** shoot at targets that do not exist? Absolutely, indirect fire from artillery may not be fired directly at a target. Indirect fire may be shot to an X-Y location. Therefore, the SQL statement must count shots against null targets.

5. In order to show the output in a meaningful table, group the results by REP, MUNITION, or TTUNIT, etc. Think of the output as a table with row headings to identify the results. The row headings are defined by the field name in the SELECT statement. Combine the results for each REP by using the GROUP BY SQL command. Grouping by REP will produce output as "1, 2, 3, 4, 5, 6..." in the REP column. Display the results in a meaningful order using the ORDER BY command. The following two examples show the SELECT, GROUP BY, and ORDER BY combinations for a query that sorts by REP and a query that sorts by MUNITION and TTUNIT. GROUP BY must be placed before the ORDER BY.

```
SELECT REP, SUM(RPB) as SHOTS  
GROUP BY REP  
ORDER BY REP;  
SELECT MUNITION, TTUNIT, SUM(RPB) as SHOTS  
GROUP BY MUNITION, TTUNIT  
ORDER BY MUNITION, TTUNIT;
```

Shots Query SQL Statement Example

The previous SQL examples (SELECT, FROM, WHERE, GROUP BY, ORDER BY) are combined:

```
SELECT MUNITION, TTUNIT, SUM(RPB) as SHOTS  
FROM [EA Summary]
```

```

WHERE (TTUNIT IS NULL OR (TTUNIT IN (List of Targets))) AND
(MUNITION IS NOT NULL AND MUNITION IN (List of Munitions))
AND RPB > 0
GROUP BY MUNITION, TTUNIT
ORDER BY MUNITION, TTUNIT;

```

Shots Query Output Table Example

Table 5-2 Shots Query Output Table Example		
MUNITION	TTUNIT	SHOTS
Mun_1	Target_1	6
Mun_1	Target_2	12
Mun_1	Target_3	15
Mun_2	Target_4	75
Mun_2	Target_5	16
Mun_3		6

Hits Queries

In order to count the number of hits in an EA Summary file, the user needs to define what constitutes a valid hit. It helps to define the criteria for a valid hit in English before translating them into an SQL statement.

Valid hits occur when all of the following conditions are met:

1. A hit occurs against a target. The shot does not necessarily have to be directed at the target.
2. The Query must count the total occurrences of hits. The same munition cannot hit the same target multiple times.
3. The user must be able to select specific firers, munitions, and targets.
4. The information must be organized and totaled by specific criteria like "totals for each munition against each target" or "totals for each replication".
5. The EA Summary file identifies a hit against a vehicle turret or hull with "HIT-T" and "HIT-H", respectively. All other items in the HM column are considered misses.

```

WHERE HM IN ("HIT-T", "HIT-H")
WHERE HM = "HIT-T" OR HM = "HIT-H"

```

Note: Two versions of the hit criteria are shown here. SQL accepts both formats. The first statement is read as "Where HM values match values IN the following list ("HIT-T" or "HIT-H"). The second statement is more intuitive, however this example shows that a list of two items is slightly longer than the first method. When searching for many criteria, the length of the SQL string is much shorter using the first notation.

These five conditions define a valid shot. Now they need to be translated into SQL:

1. Calculations are defined in the SELECT statement. To count the number of hits, include this statement in the query.

```

SELECT COUNT(HM) as HITS

```

- Request information on specific firers, munition, targets, or units in the "Where" statement.

```
WHERE (TTUNIT IN (List of Targets)) AND (MUNITION IS NOT NULL AND MUNITION IN (List of Munitions))
```

- In order to show the output in a meaningful table, group the results by REP, or MUNITION, or TTUNIT, etc. Think of the output as a table with row headings to identify the results. The row headings are defined by the field name in the SELECT statement.

```
SELECT REP, COUNT(HM) as HITS
GROUP BY REP
ORDER BY REP;
```

Hits Query SQL Statement Example

```
SELECT REP, COUNT(HM) as HITS
FROM [EA Summary]
WHERE (TTUNIT IN (List of Targets)) AND (MUNITION IS NOT NULL AND MUNITION IN (List of Munitions)) AND (HM = "HIT-T" OR HM = "HIT-H")
GROUP BY REP
ORDER BY REP;
```

Hits Query Output Table Example

Table 5-3 Hits Query Output Table Example	
REP	HITS
1	35
2	42
3	31
4	45
5	57

System Kills Queries

System kills are kills against non-personnel targets. This may include all vehicles, armor, or artillery. There is no way to identify whether a target is a system or personnel in the EA Summary file. Therefore, when creating the query, the user must select which items are systems by limiting the list of targets. **TRAC HEAT** provides a method of creating unit tables that define target categories to simplify the target selection process.

A valid system kill occurs when the following conditions are satisfied:

- The target is a system.
- The kill type (mobility, catastrophic, fire power, etc) must be specified and counted. Overkill of the same system object must be ignored.
- The user must be able to select specific firers, munitions, and targets.
- The information must be organized and totaled by specific criteria like "totals for each munition against each target" or "totals for each replication".

These requirements are translated into SQL:

- Select only the TTUNIT Codes associated with systems or use a subquery that calls a unit table that pre-defines the systems.


```
WHERE (TTUNIT IN (List of Targets))
WHERE (TTUNIT IN (SELECT Unit FROM [UNITS-Sample-Red] WHERE ([System] = -1))
```

The second example shown above uses a subquery in SQL. A subquery will define a set of criteria by looking up values in another table. The sample red units table in **TRAC HEAT** (UNITS-SAMPLE-RED) contains information on which units are system objects. In this example, the subquery would return (*List of Targets*).

Subqueries have three main advantages.

- With lists of items, the subquery text is shorter.
- The user does not need to remember all system types or unit IDs each time a query is created.
- When the scenario changes, the unit table in the subquery must change but the different unit IDs will not matter.

2. Select the applicable kill types. Since kill types are text, the query cannot sum them. Kill types must be counted.

```
SELECT COUNT(KILL) as SysKILLS
WHERE (KILL IN ("F", "K", "M", "MF"))
```

3. Request information on specific firers, munition, targets, or units in the WHERE statement.

```
WHERE (MUNITION IN (List of Munition))
```

4. In order to show the output in a meaningful table, group the results by REP, or MUNITION, or TTUNIT, etc. Think of the output as a table with row headings to identify the results. The row headings are defined by the field name in the SELECT statement.

```
SELECT MUNITION, TTUNIT, COUNT(KILL) as SysKILLS
GROUP BY MUNITION, TTUNIT
ORDER BY MUNITION, TTUNIT;
```

System Kills Query SQL Statement Example

```
SELECT MUNITION, TTUNIT, COUNT(KILL) as SysKILLS
FROM [EA Summary]
WHERE (TTUNIT IN (List of Targets) AND (MUNITION IN (List of Munitions)) AND (KILL IN ("F", "K", "M", "MF")))
GROUP BY MUNITION, TTUNIT
ORDER BY MUNITION, TTUNIT;
```

System Kills Query Output Table Example

Table 5-4 System Kills Query Output Table Example		
MUNITION	TTUNIT	SysKILLS
Mun_1	Target_1	6
Mun_1	Target_2	12
Mun_1	Target_3	15
Mun_2	Target_4	75
Mun_2	Target_5	16
Mun_3		6

Personnel Kills Queries

The EA Summary file has a PRNKILL column that totals the number of personnel killed with each hit. Calculating the total is simply a matter of using the SUM function on that column. A valid personnel kill occurs when the following conditions are satisfied:

1. The target is dismounted personnel, mounted personnel, or system crew.
2. The kill type (mobility, catastrophic, fire power, personnel, etc.) must be specified and counted. Overkill of the same system is ignored.

3. The user may select specific firers, munitions, and targets.
4. The user may select mounted units, dismounted units, or crew.
5. The information must be organized and totaled by specific criteria like "totals for each munition against each target" or "totals for each replication".

These requirements are translated into SQL:

1. Sum the number of personnel killed. Shots or hits are not checked since the personnel kills for a miss is zero.

```
SELECT SUM(PRNKILL) as PersKILLS
```
2. Since personnel can be killed when on a vehicle, the kill type is not limited to just "P". "K" kills also refer to kills of an entire squad.

```
WHERE (KILL IN ("F", "K", "M", "MF", "P"))
```
3. Request information on specific firers, munition, targets, or units in the "Where" statement.

```
WHERE (TTUNIT IS NULL OR (TTUNIT IN (List of Targets))) AND  
(MUNITION IN (List of Munitions))
```
4. When personnel units are killed while mounted on a vehicle, **CASTFOREM** lists the target as the vehicle, not the personnel unit that was killed. Therefore, it may be necessary to identify whether a unit is dismounted or mounted when looking for personnel kills. Add the following SQL code to the WHERE statement.
 - Mounted Only: TCUNIT IS NOT NULL
 - Dismounted Only: TCUNIT IS NULL
 - Either Mounted or Dismounted (all): Do not include TCUNIT phrase
 - Specific Carriers: TCUNIT IN (List of Carriers)
5. In order to show the output in a meaningful table, group the results by REP, or MUNITION, or TTUNIT, etc. Think of the output as a table with row headings to identify the results. The row headings are defined by the field name in the SELECT statement.

```
SELECT REP, SUM(PRNKILL) as PersKILLS  
GROUP BY REP  
ORDER BY REP;
```

Personnel Kills Query SQL Statement Example

```
SELECT REP, SUM(PRNKILL) as PersKILLS  
FROM [EA Summary]  
WHERE (TTUNIT IN (List of Targets)) AND (MUNITION IN (List of  
Munitions)) AND (KILL IN ("F", "K", "M", "MF", "P"))  
GROUP BY REP  
ORDER BY REP;
```

Personnel Kills Query Output Table Example

Table 5-5 Personnel Kills Query Output Table Example	
REP	PersKILLS
1	35
2	42
3	31
4	45
5	57

Range Queries

Generating a "Range" query is identical to the queries above with minor modifications. There are basically two types of range queries used in **TRAC HEAT**. The first type finds the average range of shots, hits, or kills for a specific munition. The second type of range query involves totaling the shots, hits, or kills over a specified range.

1. To calculate the average range of a shots, hits, or kills query, change the `SELECT` statement to include the `AVG` function instead of the `SUM` or `COUNT` function. This example shows the shots query example modified for the average range:

```
SELECT MUNITION, TTUNIT, AVG(FIRRNG) as RANGE
FROM [EA Summary]
WHERE (TTUNIT IS NULL OR (TTUNIT IN (List of Targets))) AND
(MUNITION IS NOT NULL AND MUNITION IN (List of Munitions))
AND RPB > 0
GROUP BY MUNITION, TTUNIT
ORDER BY MUNITION, TTUNIT;
```

2. To calculate the number of shots, hits or kills over a specified range, simply add the conditions to the "Where" statement. The following example shows the "Hits" query example modified for a range from 500 to 1000 meters. Use `<`, `>`, and `=` to indicate boundaries in the query range.

```
SELECT REP, COUNT(HM) as HITS
FROM [EA Summary]
WHERE (TTUNIT IS NULL OR (TTUNIT IN (List of Targets)))
AND (MUNITION IN (List of Munitions)) AND (HM = "HIT-T" OR
HM = "HIT-H") AND (FIRRNG > 500 AND FIRRNG <= 1000)
GROUP BY REP
ORDER BY REP;
```