



Apelon DTS™

TQL 4.3 Release Notes

June 2015

Apelon

Apelon Inc
750 Main Street
Suite 1500
Hartford, CT

P 203.431.2530
www.apelon.com

Table of Contents

TQL V4.3 Release Notes	4
DESCENDANT Keyword	4
DTSLayout.....	4
Bug Fixes	4
TQL V4.2 Release Notes	5
FOR Statement.....	5
Anonymous Attributes	6
TQL V4.1 Release Notes	7
Block Structuring	7
Conditional Statements	8
Extended SET	9
Extended LOG and PRINT	9
New TQL Attributes	10
Query Header	10
Escapes.....	11
TQL V4.0 Release Notes	12
Installation.....	12
Query Structure	12
Attribute Delimiter.....	12
Term Collections.....	12
Selectors.....	13
User Variables.....	13
Expressions	13
User Functions	14
Contexts	14
TQL Variables	15
TQL Attributes.....	15
TQL Commands.....	16
TQL Menus	18

Output Panel.....	18
TQL Batch Command.....	18

TQL V4.3 Release Notes

DESCENDANT Keyword

The DESCENDANT keyword has been added. This keyword represents all the (recursive) subconcepts/children of the referent Concept. For Thesaurus Namespaces, the AXIS association is used.

DTSLayout

TQL 4.3 implements the new DTS V4.3 Module Architecture features related to Layouts.

Bug Fixes

TQL 4.3 fixes a number of bugs, including now correctly wrapping Concepts dropped into the query panel with quotes.

TQL V4.2 Release Notes

FOR Statement

TQL V4.2 introduces the FOR statement, which iterates a Concept or Term object over the instances of an attribute. The statement is only available in a Collection and cannot be nested: a FOR statement cannot occur within the scope of another FOR statement.

A FOR statement consists of a FOR attribute and an associated *statement-block*:

```
FOR for-attr statement-block
```

The *for-attr* is any single TQL or DTS attribute that can have multiple values, e.g., PROPERTY, SYNONYM, a Property Type, Association Type, etc.

The purpose of the FOR statement is to allow processing of specific attribute instances. TQL filtering operations such as *selectors* (WITH ...) and *conditional statements* (IF ...), have an “if any” interpretation: if any of the attributes meet the conditions, the Concept or Term is passed to the associated *statement-block* with all of its attributes. This makes it impossible to perform operations on specific instances of an attribute such as only printing a Synonym attribute if its value begins with “A”.

The FOR statement enables attribute-specific operations by passing a Concept or Term object to the associated *statement-block* that only contains one instance of the specified attribute:

```
FROM [^SNOMED CT^] WITH NAME EQUALS "myocardial infarction
(disorder)"

FOR SYNONYM

    IF SYNONYM EQUALS "MI*" PRINT NAME, SYNONYM;
```

The third line of this query is executed once for each Synonym value on the selected Concept. If the Term name of any of these Synonyms begins with “MI”, it is printed.

As shown above, the FOR statement is usually used in conjunction with a *conditional-statement* to perform attribute filtering.

The *delete-attributes-statement*, *set-attributes-statements* and *update-attributes-statement* can also be used with the FOR statement to operate on specific instances of attributes:

```
FROM [Demo]

FOR ^My Prop^

    IF ^My Prop^ EQUALS "old*" DELETE ^My Prop^;
```

Only instances of My Prop whose value begins with “old” will be deleted.

Anonymous Attributes

TQL V4.2 also adds *anonymous attributes* to facilitate the processing of query contexts. In previous versions of TQL, non-namespace-qualified *attribute* references of the form `^Code in Source^` implicitly refer to the named *attribute* found in the “default” namespace: the namespace specified in the enclosing *collection context*. This *attribute* form was not permitted in non-namespace-specific *contexts* such as the All Namespace *context* (`FROM ALL WITH ...`) or the Subset, ConSet or TermSet *contexts*.

In Version 4.2, the form `^Code in Source^` can be used in these contexts. When the current *context* is an All Namespace, Subset, ConSet, or TermSet *context*, a non-namespace-qualified *attribute* is called an *anonymous attribute*. The type (actual instance) of the *anonymous attribute* is not resolved until run-time and is determined relative to the namespace of the current concept (or term) in the *collection*. For each selected concept in these *statements*:

```
FROM ALL WITH NAME EQUALS "XY*" EXPORT NAME, ^Code in Source^;
```

```
FROM {MySubset} EXPORT NAME, ^Code in Source^;
```

the existence of a `Code in Source` Concept Property Type from the concept’s namespace is determined, and if present, its value on the concept is exported.

Since TQL cannot know the attribute type (Concept Property, Term Association, etc.) of an *anonymous attribute*, *anonymous attributes* usually have a type code specified: `^My Map Association^(CA)`. (See the **TQL Editor User Guide** for a complete list of available type codes.) If no type is specified, Concept Property Type is used as shown in the above examples.

Anonymous attributes can be used in *expressions* in *export-statements* and *output-statements* (LOG and PRINT), and *predicates*. Use in *expressions* includes the displaying (`->`) form. The examples below demonstrate these use cases:

```
FROM ALL WITH ^My Map[Demo]^ EXISTS
```

```
    EXPORT NAME, NAMESPACE, ^My Map[Demo]^, ^My Map[Demo]^->^Code in Source^;
```

```
FROM ALL WITH NAME EQUALS "xy*"
```

```
    IF ^Code in Source^ EQUALS "1*"
```

```
        EXPORT NAME, NAMESPACE, ^Code in Source^;
```

Anonymous attributes CANNOT BE USED in selectors.

TQL V4.1 Release Notes

Block Structuring

Certain TQL statements, in particular the Collection Statement, now accept multiline Statement Blocks in their syntax. This is most easily described in an example:

```
FROM [^MySpace^] with CONCEPT_NAME EQUALS "X*" {
    SET ^Code in Source^ = "C" & CONCEPT_ID;
    SET INV ^Parent Of^ = "MyRoot";
    LOG CONCEPT_NAME & " updated";
}
```

The Statement Block is the three statements enclosed in brackets. The block is an intrinsic part of the Collection Statement. Single statement “blocks”, without brackets, are still supported.

In the above example, the statements in the block are executed once, in the order presented, for each Concept in the specified Collection, i.e., the Concepts in MySpace whose names begins with “X”. Generally, the same result could be achieved by three separate Collection Statements without a block, but the use of the block is more efficient and there are examples where the side effects of a statement make blocks more natural and clear. This is especially true when Conditional Statements (see below) are used.

Not all TQL statements can be used in a statement block. More generally, certain statements may only be used in specific locations in a query or in conjunction with other statements. A non-contextual statement can only appear at the top level of a query. A contextual, or subordinate, statement, in turn, can only be used in the scope of a *collection-statement*. Contextual statements themselves can be restricted or un-restricted. A restricted contextual statement must be the **ONLY** statement in the scope of a *collection-statement*. A restricted contextual statement cannot occur in a statement block. An unrestricted contextual statement can occur anywhere in the scope of its enclosing *collection-statement* and appear at any nested (block) level. Some statements can be used in multiple positions, often dependent on the statements’ arguments. The table below summarizes these dependencies. See **The TQL User Guide** for more detailed descriptions of statement types and classifications.

Statement	Classification	Notes
<i>collection-statement</i>	Non-contextual	
<i>conditional-statement</i>	Unrestricted contextual	
<i>constrain-statement</i>	Non-contextual	

<i>create-context-statement</i>	Non-contextual	
<i>delete-context-statement</i>	Non-contextual	
<i>edit-statement</i>	Contextual	Restricted and unrestricted forms
<i>export-statement</i>	Restricted contextual	Some <i>collection</i> restrictions
<i>output-statement</i>	Contextual and non-contextual	Arguments must be consistent with statement usage
<i>parameter-statement</i>	Non-contextual	
<i>set-variables-statement</i>	Non-contextual	Arguments can be used in <i>set-attrs-statement</i> .

Conditional Statements

TQL 4.1 introduces the conditional statements IF, ELSE and ELSEIF. The IF statement, only available in a Collection, takes a single **predicate** argument. A predicate is similar to a Selector: it has an LHS that is an attribute, a TQL operator, and a RHS Expression. The predicate is evaluated for each Concept (or Term) in the enclosing Collection. If the predicate evaluates to true, the subsequent statement (or statement block) is executed. If the predicate evaluates to false, the following statement(s) is ignored.

```
FROM [Demo]
```

```
  IF NAME EQUALS "T*"
```

```
    PRINT NAME;      --print concepts in Demo that begin with 'T'
```

In the above query, the equivalent result could be accomplished with a Selector. In fact, the use of a Selector is more efficient than a conditional since the conditional (IF) predicate must be evaluated for every object in the Collection. Once a Collection has been created, however, the IF statement can provide new functionality not previously available in TQL.

The ELSE statement must immediately follow an IF (or ELSEIF) and inverts the sense of the preceding predicate; if the preceding IF (or ELSEIF) predicate evaluated to false, the statement (or statement block) following the ELSE is executed.

The ELSEIF statement, which must immediately follow an IF or another ELSEIF, combines the function of an ELSE and IF to more easily permit “chaining” of conditionals. Without this form, nested blocks would be required.


```

FROM [^States of the Union^] {
    IF Capital equals "a*" PRINT "The Capital of "&NAME&" starts with
A";
    ELSEIF Capital equals "b*" PRINT "The Capital of "&NAME&" starts
with B";
    ELSEIF Capital equals "c*" PRINT "The Capital of "&NAME&" starts
with C";
    ELSE PRINT "The Capital of " & NAME & " is not an A, B, or C";
}

```

Predicate syntax is different from Selectors in two ways. First, the LHS of a predicate can be a user or TQL variable, so the values of these elements can be tested. Second, the RHS expression can be contextual (contain a DTS or TQL attribute) that is evaluated in the context of the Collection.

```

FROM [Demo]
    --test if each concept's Code is the same as its Code in Source
    IF CODE NOT_EQUALS ^Code in Source^ PRINT NAME & "Is
mismatched.";

```

Extended SET

The SET statement now accepts mixed attribute and variable arguments in any context:

```

SET %COUNT = 0;
FROM [Demo] SET ^Demo Prop^ = "foo", %COUNT = %COUNT+1;
LOG %COUNT&" properties set";

```

Extended LOG and PRINT

The LOG and PRINT statements now use the same argument processor as EXPORT, so contextual, encoded and displaying forms can be used (when inside a Collection statement, of course). The only difference between PRINT and EXPORT is that because of availability of the SORTED_BY modifier, EXPORT cannot be used in a block, and PRINT does not produce an automatic header line.

New TQL Attributes

TQL 4.1 introduces a number of new Attributes. The NAME, CODE, ID, and STATUS attributes are simply shorthand for CONCEPT_NAME and TERM_NAME, CONCEPT_CODE and TERM_CODE, CONCEPT_ID and TERM_ID, and CONCEPT_STATUS and TERM_STATUS. TQL is able to distinguish between the Concept/Term interpretations of these variables based on the enclosing statement Context (or Expression context). This automatic interpretation was already present in the NAMESPACE TQL attribute.

The NAMESPACE TQL variable can now be used in Collection Selectors. This variable could be useful when processing multi-Namespace contexts such as Subsets or ConSets.

The PROPERTY, ROLE, and ASSOCIATION attributes have been added to support access to all instances of these attribute types in EXPORT, PRINT, and LOG statements. PROPERTY, ROLE, and ASSOCIATION can be used in a similar manner to the SYNONYM attribute. The query below will export all property values on the Concepts in States of the Union:

```
FROM [^States of the Union^] EXPORT NAME, PROPERTY;
```

Since only the value of the Properties will be exported in the above, the new TYPE modifier is available to access the type name of any SYNONYM, PROPERTY, ROLE or ASSOCIATION instance. The VALUE modifier is also available which returns the string value of the attribute. A more complete query would then be:

```
FROM [^States of the Union^] EXPORT NAME, PROPERTY.TYPE,  
PROPERTY.VALUE;
```

The INV and DEF prefixes can also be used in appropriate contexts as shown below:

```
FROM [^TRIAD^] EXPORT NAME, DEF ROLE.TYPE, DEF ROLE;  
  
FROM/TERMS [^States of the Union^]  
  
EXPORT NAME, INV SYNONYM.TYPE, INV SYNONYM.VALUE;
```

Query Header

All TQL Queries are now saved with an initial “header” comment line that gives the TQL version, user and saved date/time:

```
/* TQL V4.0 Query saved by dtsadmin on 2 Sep 2014 21:08:12 */
```

Display of the header line in the query editor area can be turned on or off by a new Show Header Line item in the File|Preferences menu.

Escapes

Finally, TQL 4.1 supports escaping the caret (“^”) so that this character can appear in attribute literals:

```
From {^My\^Subset^} print name, ^Special\^Property^;
```

TQL V4.0 Release Notes

TQL Version 4.0 is a major update to TQL for DTS Version 4. TQL V4 adds features that support new DTS capabilities such as versioning and also enhances TQL with new constructs. TQL V4.0 is the first in a series of TQL updates that will add significant new reporting and editing capabilities. This document summarizes the specific additions in Version 4.0. For further details on TQL V4.0 see the **TQL User Guide**, **TQL How-To Guide** and **TQL Reference Guide**.

Installation

Installation of TQL 4.0 follows the standard DTS Editor Plug-in process: unzip the installation kit into your DTS installation folder. If you have a previous version of TQL installed, you should also remove TQLUtils.jar from `lib\modules`. This jar is no longer used; all TQL-specific code now resides in a single jar.

Query Structure

Attribute Delimiter The most visible change in V4.0 is a modification in the delimiter used for TQL Attributes (Namespaces, Subsets, Authorities, Conssets, TermSets, Concepts, Terms, and DTS Concept/Term Attributes). In order to accommodate more general argument expressions (see below), TQL Attributes are now delimited using the caret (a.k.a. up-arrow) character: “^”.

```
FROM [^SNOMED CT^] EXPORT concept_name, ^Code in Source^;
```

As in previous versions of TQL, use of the attribute delimiter is only required when the TQL Attribute name includes a punctuation character other than underscore.

TQL V4 can automatically convert older queries to V4 format. See **TQL Menus** below for further information.

Term Collections TQL V4.0 supports the creation of *Collections* based on Terms as well as Concepts. Term *Collections* can be used in *Export* statements (along with Term attributes) and to build local *TermSets* (a new V4.0 feature complementary to *ConSets*). To create a Term *Collection* use the **TERMS** modifier on the Collection Statement:

```
FROM/TERMS [^SNOMED CT^]
    WITH TERM_NAME EQUALS "a*" EXPORT TERM;
```

Term support extends to cross-type attribute expressions that use the “pointing” syntax on synonyms:

MySynonym->TermProp	valid in a Concept <i>Collection</i> ; returns the value of the <code>TermProp</code> Property on the synonym’s term
---------------------	----------------------------------------------------------------------------------------------------------------------

Inv MySynonym->ConProp valid in a Term *Collection*; returns the value of the ConProp Property on the synonym's concept

Selectors

Concept (and Term)-based selectors, e.g. CONCEPT EQUALS, ^Assn^ EQUALS, CONCEPT CHILD_OF, now accept namespace-qualified expression values. This facilitates use of Concept/Term Drag and Drop from other DTS panels and use of TQL Parameters in the expression.

... WITH CONCEPT EQUALS "North[States of the Union]" ...

... WITH ^Map to ICD^ EQUALS @TARGET@ ...

User Variables

In addition to TQL Variables such as AXIS and DELIMITER, TQL V4 supports User Variables. User Variables are names that begin with the percent ("%") character. Values are assigned to User Variables in the SET command and can be used in *Expressions* (see below).

```
SET %NAME = "Jack";
```

Like TQL Variable names, User Variable names are case-insensitive.

Expressions

TQL syntax previously called for "value" elements such as arguments in *selectors*, and EDIT, SET and EXPORT statements, to be numeric or string literals. In TQL V4.0, these elements are now instances of the *Expression* element. An *Expression* is a sequence of *Expression Elements* separated by expression operators. An Expression Element can be a parameter (@PARAM@), numeric literal (12.34), string literal ("required"), a TQL Variable (AXIS), a User Variable (%NAME), a TQL Attribute (CONCEPT_NAME), a DTS Attribute (^Code in Source^), a TQL Function (LENGTH) or a User Function (see below).

The expression operators and their semantics are:

The plus sign ("+") Takes the numeric head of each of its arguments and returns the numeric sum (as a string).

The minus sign ("-") Takes the numeric head of each of its arguments and returns the result of subtracting the second value from the first (as a string).

The ampersand("&") Returns the concatenation of the two string values.

The value of an *Expression* is the result of applying each expression operator to the values of its right and left *Expression Elements*. Expression evaluation proceeds strictly left to right (grouping via parentheses is currently not supported). Thus:

```
SET %n = 12 + "34AB" & 56;
```

```
LOG %n;
```

prints “ 4656” on the console.

The primary restriction on *Expression Elements* is that there can only be one instance of a “contextual” element (an element based on a Concept or Term), e.g. a TQL Attribute, DTS Attribute or TQL Function, in an *Expression*.

```
EXPORT "Capital of" & ^State Name^, "is" & ^State Capital^;
SET ^Code in Source^ = "CIS" & concept_code;
```

Certain TQL statements have additional restrictions on *Expressions*. These are documented in the associated statement descriptions in the **TQL User Guide**.

User Functions

In addition to the standard TQL Functions (COUNT and LENGTH), TQL V4 supports *User Functions*. User Functions take a TQL or DTS Attribute as an argument and return an associated string value. For example, the %REVERSE(arg) function might return the string value of its argument with the characters reversed. A User Function is created by writing a Java class that extends the `TQLFunction` class. This class, packaged in a jar file and placed in lib\modules, is recognized by TQL on start-up and can subsequently be used in *Expressions* in the same way as a TQL Function.

Contexts

Namespace and Subset *Contexts* have been extended to support specification of Versions and snapshot Dates. The new extended Namespace *Context* syntax is:

```
[MySpace:VersionName] and
[MySpace#Date]
```

where `VersionName` is string literal containing the name of a Version in the Namespace, and `Date` is a string literal containing a date. A number of standard date formats are available.

```
[^SNOMED CT^:"2013.07.13AA"]
```

```
[^SNOMED CT^#"12/25/2013"]
```

Subsequent operations that incorporate the *Context*, like an EXPORT, will use Concept snapshots from the associated Version/Date. Subset *Context* forms are similar. To simplify query creation, Version names are accessible (and droppable) from the TQL Attribute Chooser.

TQL now also supports the creation and deletion of Namespace and Authority *Contexts*, subject to DTS V4 Permission considerations. See **TQL Commands** below for details.

Finally, Authority names are available in the TQL Attribute Chooser.

TQL Variables

The following changes to TQL Variables have been made:

AXIS	<p>The syntax and semantics of the AXIS variable have changed. First, specification of an inverse relationship is now part of the value string. Thus, to use a “Part Of” association as the hierarchy axis, you would use:</p> <pre>SET AXIS = "inv Part Of";</pre> <p>(Inverse is required since this association points the “wrong way” for hierarchical display.) The inverse designator, “inv” is case insensitive, but the name of the association (or role) is case sensitive.</p> <p>Second, AXIS can now be used with Ontylog (and Ontylog Extension) Namespace <i>Collections</i>. The default value of AXIS is the empty string, “”. This corresponds to a hierarchical relationship of Superconcept/Subconcept for Ontylog Namespaces and a “Parent Of” association for Thesaurus Namespaces. Any non-empty value of AXIS will cause the association, or role, named by the AXIS value to be taken as the hierarchical relationship for any subsequent <i>Collection</i> operations independent of Namespace type. To disambiguate between similarly named roles and associations, TQL attribute type designators can be used in the AXIS value:</p> <pre>SET AXIS = "inv Part Of(R)";</pre>
DATE_FORMAT	<p>A new TQL Variable whose string value specifies how dates are output in EXPORT, PRINT and LOG statements. The format value is the same as that used by the Java <code>SimpleDateFormat</code> class pattern. The TQL default is “dd-MMM-YYYY”: “25-DEC-2013”.</p>
TIME_FORMAT	<p>A new TQL Variable whose string value specifies how times are output in EXPORT, PRINT and LOG statements. The format value is the same as that used by the Java <code>SimpleDateFormat</code> class pattern. The TQL default is “h:mm a”: “12:08 PM”.</p>
SIZE	<p>A new read-only TQL Variable that holds the size (number of Concepts or Terms) of the last <i>Collection</i> (FROM ... WITH ...) operation.</p>

TQL Attributes

The following new TQL Attributes are available:

CONCEPT_STATUS	The Status of a Concept snapshot. Values are “ACTIVE”, “INACTIVE” and “DELETED”.
TERM	The Term itself.

TERM_NAME	The Name of a Term.
TERM_CODE	The Code of a Term.
TERM_ID	The ID of a Term.
TERM_STATUS	The Status of a Term snapshot. Values are “ACTIVE”, “INACTIVE” and “DELETED”.
VERSION_NAME	The name of the Version associated with a Concept or Term snapshot.
VERSION_DATE	The release date of the Version associated with a Concept or Term snapshot.

TQL Commands

The following new TQL Commands and/or Command options are available:

FROM/TERMS ... Designates that the resulting *Collection* should be Term-based. The associated *Collection Context* must be a Namespace or TermSet. Additional restrictions on this form apply, see the **TQL User Guide** for further information.

FROM/TERMS WITH ... DELETE_TERMS; Deletes (sets to Inactive) all the Terms in the Collection. See also the PERMANENT modifier below.

FROM/STATUS="INACTIVE" ... Specifies that the Concepts or Terms in the resulting *Collection* must have the designated status. Allowable values for the STATUS modifier are: ALL, ACTIVE, INACTIVE, and DELETED. The default is ACTIVE. The STATUS modifier can be combined with the TERMS modifier.

FROM [MySpace] WITH ... UPDATE CONCEPT_NAME = ^CONCEPT_CODE^+"concept";

FROM/TERMS [MySpace] WITH ... UPDATE TERM_NAME = ^TERM_CODE^+"term";

The UPDATE command supports updating the CONCEPT_NAME, CONCEPT_STATUS, TERM_NAME and TERM_STATUS TQL Attributes. The RHS of the command is a, potentially contextual, *Expression*. For STATUS, the resolved *Expression* value must be “ACTIVE”, “INACTIVE”, or “DELETED” (case insensitive). Setting/updating of other Term Attributes (Term Properties, Term Associations, and Inverse Synonyms) are supported in the second form (using a Term-based *Collection*).

CREATE [MySpace:MyAuthority]; Creates the specified Namespace with the given Authority. Note that all creation and deletion statements check for presence of the required User Permission.

DELETE [MySpace]; Deletes the specified Namespace.

CREATE {MySubset:MyAuthority} FROM ... Creation of Subsets requires specification of an Authority in TQL V4.0.

CREATE <MyAuthority>; Creates the specified Authority.

DELETE <MyAuthority>; Deletes the specified Authority;

FROM [MySpace] DELETE_CONCEPTS/PERMANENT;

For DTS V4, deletion of Concepts and Terms through the default DELETE_CONCEPTS and DELETE_TERMS commands set the status of the selected Concepts and Terms to Inactive; the Concepts and Terms ARE NOT permanently deleted. Concepts and Terms CAN be permanently deleted, however, through the use of the PERMANENT modifier as shown above. All forms of the DELETE_CONCEPTS and DELETE_TERMS statements support this modifier. PERMANENT is also available on the DELETE attribute statement for permanently deleting Terms associated with deleted Synonyms. The PRUNE_TERMS modifier must also be used.

FROM [MySpace] EXPORT_NAMESPACE; This command is now the recommended command for exporting Namespace data. The export file contains all TypeDefs and Attributes of the Namespace, including those for Namespace and Version Properties. New XSDs have been written for all XML exports and are provided in the kit. Loading is supported by Import Wizard V4.1.

FROM/STATUS="ALL" [MySpace] EXPORT_NAMESPACE;

EXPORT_NAMESPACE (and EXPORT_CONCEPTS) support the STATUS= modifier. Only Concepts and Terms having the designated status are included in the export. Only ACTIVE Concepts are exported if STATUS is not specified. The value of STATUS is included in Concept and Term elements and setting of STATUS on load is supported by Import Wizard V4.1.

FROM {MySubset} EXPORT_SUBSET; This command has new semantics. The previous EXPORT_SUBSET functionality is provided by EXPORT_CONCEPTS/SUBSET_VIEW (see description below). EXPORT_SUBSET now provides a complete extract of Subset information including Subset description, expression, and TypeDefs and Attributes for Subset and Subset Version Properties. Loading is supported by Import Wizard V4.1.

FROM {MySubset} EXPORT_CONCEPTS/SUBSET_VIEW This command/modifier provides the same functionality as the previous EXPORT_SUBSET command. Information on the Concepts participating in the Subset are

exported, using the Subset-centric view, e.g., Subset hierarchy rather than natural hierarchy.

`PARAMETER @PARM@[[:[type]][:“help string”]];` The syntax of this command has changed in V4 to include an optional explicit type. The parameter type is no longer determined by the usage context. Available types are: `STRING` (the default), `INTEGER`, `NUMBER`, `BOOLEAN`, `CONCEPT`, `TERM`, and `FILE`.

TQL Menus

The TQL File menu contains a new item, Preferences. Available options are:

Number Output Lines	When selected, lines in the TQL Output tab panel will be numbered.
Test for V3 Query	When selected, parsed queries will be scanned for non-V4 attribute delimiters. This enables the rewriting (conversion) of the query to V4 format. After parsing, non-V4 queries will show an informational Rewrite Query dialog. From this dialog, a rewrite operation can be rejected, the query can be rewritten, or the query can be rewritten and subsequent queries can be automatically rewritten (the dialog will be suppressed). To reinstate display of the dialog after suppression, unselect then reselect <code>Test for V3 Query</code> .

Note: This option should be deselected after V3 queries have been converted. Some valid V4 queries will report parse errors if V3 testing is enabled.

The state of these options is persisted in the TQL configuration file.

Output Panel

In addition to line numbering as described above, the base Concepts (or Terms) associated with each output line can be moved to other DTS Editor panels by selecting and dragging the DTS Icon at the left of each line. Multiple select/drag is supported.

TQL Batch Command

Apelon DTS Editor V4 Modules/Plug-ins incorporate a new format for batch arguments. See the **TQL User Guide** for details on direct invocation of the TQL class.