OpenQM Data Integrity Constraints

https://www.openqm.com/support/documentation/

Data Integrity Constraints provide a way to validate data before it is written to a file, making errors such as writing an external form date where internal form was expected a thing of the past. It is impossible to bypass the checks on a file where constraints are enabled.

Defining Constraints

Data Integrity Constraints can be applied to D-type dictionary items or to A/S types that do not have a correlative. Only one data defining item for each field may have constraints defined. The constraint expression is stored in field 12 of the dictionary item.

Constraint definitions are very similar to an I-type expression, returning True or False to indicate whether the data is valid.

The constraint expression can use most features of an I-type expression but cannot execute subroutines or reference data in other records with the exception of the EXISTS() function.

There are five special functions available in constraint expressions:

Name	Description
INTEGER()	Data must be an integer value. Low/high range checks can be included.
NUMBER()	Data must be numeric. Low/high range checks can be included.
STRING()	Data must be a string with min/max length check.
MATCHING()	Data must match a pattern template.
EXISTS()	Check existence of a related record.

For a multivalued field, all values must comply with the validation expression.

Examples

Event	Description
`INTEGER(1, 10)`	Range of integer values
`INTEGER()`	Integer, no range check
`INTEGER(4)`	Integer, minimum value 4
`INTEGER(, 8)`	Integer, maximum value 8
`NUMBER(0.5, 100)`	Range of numeric values
`NUMBER()`	Numeric, no range check
`NUMBER(0.5)`	Number, minimum value 0.5
`NUMBER(, 40.5)`	Number, maximum value 40.5
`STRING(1, 200)`	String, length 1 to 200 characters
`STRING(10)`	String, exact length 10 characters
`MATCHING('1-3N')`	Pattern match, 1 to 3 numeric characters
`EXISTS('SALES', @DATA)`	

Special Data Names

Name	Description
`@DATA`	Data of field being processed
`@FNO`	The field number of the field being processed
`@FNAME`	The field name of the field being processed
`@ID`	Id of record being written
`@RECORD`	Data record being written

Examples

Check that a MONTH.ENDING field is the last day of a month:

```
OCONV(@DATA + 1, "DD") = 1
```

Check that a code is 4 hexadecimal characters:

```
LEN(@DATA) = 4 AND CONVERT("0123456789ABCDEF", "", @DATA) = ""
```

Check that an expiry date is after an issue date:

```
EXPIRY.DATE > ISSUE.DATA
```

The expiry date example could be in the dictionary definition of either field and that field name could be replaced with @DATA.

Compiling Constraints

The expressions for each field to which constraints are applied are merged into a single operation using the COMPILE.CONSTRAINTS command. The compiled expression is stored in the data file.

```
> COMPILE.CONSTRAINTS MYFILE
```

Constraint Validation

The constraint expression is executed prior to any action that writes data to the file. If a pre-write trigger is defined, this is executed before the constraint check. If the expression returns False, indicating a validation error, the write operation is abandoned and the system returns an error by the first applicable of the following four actions:

- If there is an exception handler for SYS.FILESYS.CONSTRAINTS, this exception is thrown.
- If the WRITE has an ON ERROR clause, this is executed.
- If the WRITE has an ELSE clause, this is executed.
- The program is aborted.

In all cases, the @FNO and @FNAME variables will hold the field number and field name of the failing element of the data being written.

Constraint Logging

As an alternative to application failure at a validation error, the COMPILE.CONSTRAINTS command has a LOGGING option:

```
> COMPILE.CONSTRAINTS MYFILE LOGGING
```

Disabling Constraint Validation

Use of the DISABLE keyword suppresses all automatic integrity checks.

```
> COMPILE.CONSTRAINTS MYFILE DISABLE
```

In this case, no error will be reported for faulty data. Subsequent use of the VERIFY.CONSTRAINTS command will scan the file and report all records that fail validation.

```
> VERIFY.CONSTRAINTS MYFILE
```

Validating Without Writing

The QMBasic VALIDATE() function can be used to check a record before attempting to write it to the file, including when automatic checks have been disabled.

```
IF NOT(VALIDATE(FILEVAR, ID, DATA)) THEN
   DISPLAY 'Invalid data'
END
```