Rocket Uniface Library 10.4

Relational Operators

Relational operators take two operands, compare their values, and return a Boolean value (true or false). They are typically used in conditional expressions to test whether a condition is true, or not.

A conditional expression can use arithmetic expressions and relational operators. For example:

```
if (X > 3)
 message "Class not valid"
endif
if ( (1 + TAX/100) * AMOUNT <= BUDGET )
 message "Approved"
endif
```

Table: Relational Operators

Operator	Description
<	Less than
<=	Less than or equal to
!=	Not equal to
=	Equal to
==	Equal to
>=	Greater than or equal to
>	Greater than

Relational operators have a lower priority than arithmetic operators, and a higher priority than logical operators. All relational operators have the same priority, and are executed from left to right.

Arithmetic Expressions Used as Conditional Expressions

If the ProcScript compiler finds an arithmetic expression where it expects to find a conditional expression, the arithmetic expression is tested against zero or an empty string, depending on the data type. For example, the clause u_where (`X*Y`), where the fields X and Y are both Numeric, is evaluated as u_where (`X*Y != 0`).

- If an arithmetic expression is evaluated as zero, the compiler considers this to be FALSE.
- If the expression evaluates to a nonzero value, the ProcScript compilerconsiders the expression to be TRUE.

The table shows examples of this behavior:

Table: Interpreting Relational Operators

Expression	Meaning When Used as a Relational Expression
`if (X)`	`if (X!=0)`, if X is numeric
	`if (X!="")`, if X is a string

Expression	Meaning When Used as a Relational Expression
`if (1+18/100)`	`if (1+(18/100)!=0)`, that is, always TRUE

This interpretation ensures that legal but potentially confusing constructions do not cause compile-time errors due to a conflict of priorities. It is similar to the constructions used in the C programming language.

Boolean Field with Relational Operators

When using a Boolean field in a statement such as `if, while`, or `repeat`, it is not a good idea to compare the field against a specific value such as Y or 0. Instead, just use the name of the field in the expression. Uniface recognizes that it is dealing with a Boolean value and automatically performs the appropriate comparison (as defined by the packing code). This ensures that data can be moved from one DBMS to another without recoding. For example, if PAID has data type Boolean:

```
if (PAID)
 ...
endif
```

is much better than:

```
if (PAID = "Y")
 ...
endif
```

Using the field as a Boolean data type ensures that if the data is moved to a DBMS that stores Boolean values as 1 or 0, the test still succeeds (it fails if a test is made against 1 or 0).

Empty Strings with Relational Operators

In conditional expression that involve an empty string (""), only the relational operators = and != should be used. In cases where an empty string is a possibility, the ProcScript should take this into consideration, for example:

```
if ($SEARCH_PATTERN$ = "")
 read u_where (FIELD = $SEARCH_PATTERN$)
else
 read u_where (FIELD >= $SEARCH_PATTERN$)
endif
```

Numeric Data, Empty Strings, and Relational Operators

In general, when two operands are compared using a relational operator and one of those operands is numeric (that is, data type Numeric or Float), the other operand is usually converted into numeric data (for the purpose of the comparison only). There are, however, special considerations when a numeric operand is an empty string ("") or contains a space. These are summarized in the table:

Table: Null Values in Comparisons

Value of N	Logical expression			Comments
	(N="")	(N=0)	(N=" ")	
""	TRUE	TRUE	TRUE	If you prefer to have this logical expression evaluated as FALSE when the numeric operand is an empty string, use the assignment setting `$NUM_NULL_CMP_ZERO=FALSE`.
0	FALSE	TRUE	TRUE	
" "	TRUE	TRUE	TRUE	If you prefer to have this logical expression evaluated as FALSE when the numeric operand contains a space, use the assignment setting `$NUM_NULL_CMP_ZERO=FALSE`.

For more information, see $NUM_NULL_CMP_ZERO.

For example, to check for an empty string in a numeric field, N, you can use a statement such as the following:

```
if (N = "")
 ...
endif
```

This logical expression is evaluated as TRUE when N is "", and FALSE when N is 0.

Comparing Strings with Numbers

When you compare a string value with a number, if the string contains only characters used to make up numeric values (that is, `0-9`, `+`, `-`, `.`, and `e`), it is converted to a number for the comparison. The following example results in the message `"002 is greater than 1"`:

```
$1 = "002"
if ($1 > 1)
 message "%%$1 is greater than 1"
else
 message "%%$1 is less than or equal to 1"
endif
```

However, if the string contains a nonnumeric character, the result of the comparison is always FALSE. The following example results in the message `"002a is less than or equal to 1"`:

```
$1 = "002a"
if ($1 > 1)
 message "%%$1 is greater than 1"
else
 message "%%$1 is less than or equal to 1"
endif
```

To overcome this, you can use the function $number to explicitly convert the mixed-character string to a number. The following example results in the message `"002a is greater than 1"`:

```
$1 = "002a"
if ( $number($1) > 1 )
 message "%%$1 is greater than 1"
```

```
else
 message "%%$1 is less than or equal to 1"
endif
```