Converting between data types

Last Modified on 01/06/2019 4:00 am AEST

Implicit and explicit conversion

A calculation or sub calculation with a particular result type will often need to be used in a place that requires a different data type.

For example, a record field that contains a date, or number, may need to be combined with other text in order to generate an email message. In this example, the date sub calculation needs to first be converted to text.

There are two ways that conversion may occur between types.

Implicit conversion

For many data type conversions, the meaning of the data conversion is clear, and there will be no loss of information. In such cases, conversion will happen automatically.

In the following calculation, the **decimal** 0.5 is being added to the whole **number** 120. In this case, the whole number would be implicitly converted to a decimal before being added to the other decimal, and further returning a decimal result data type.

```
0.5 + 123
```

Explicit conversion

For some data type conversions, a way of converting the data may be imagined, but it may not be the kind of conversion that one wants to perform accidentally. In these cases, conversion must be explicitly performed using the **convert** function.

In the following example, the decimal 5.1 is being passed to the second parameter of the left function, which only accepts whole numbers. On its own, this will cause a calculation error to be shown because the data types do not match, and decimal does not implicitly convert back to **int**. However, a conversion can be imagined - namely to round the decimal. The convert function can be used to explicitly force this.

```
left( "Welcome", 5.1 )  -- error
left( "Welcome", convert( int, 5.1 ) )  -- OK
```

Permitted conversions

The following table shows what type conversions are implicitly and explicitly allowed.

★ **Implicit** conversion allowed from the type on the left to the type on the top

⭐ **Explicit** conversion allowed from the type on the left to the type on the top

	string	int	decimal	currency	date	time	datetime	record	bool
string	n/a	⭐	⭐	⭐	⭐	⭐	⭐		⭐
int	★	n/a	★	⭐					⭐
decimal	★	⭐	n/a	★					⭐
currency	★	⭐	★	n/a					
date	★				n/a		★		
time	★					n/a	★		
datetime	★				★	★	n/a		
record	★							n/a	⭐
bool	★								n/a

Specific conversions

Converting from record to string

A record will implicitly convert to a string. When this is done, then name of record is returned.

For example, the following calculation will retrieve the name field of each record, and prefix it with the text provided.

```
'Hello ' + all(Person)
```

Converting between record objects

If a result data type is **record**, then the record object is also tracked.

Record calculations can be *implicitly* converted from an object to a parent object. For example, an expression that returns a list of Employees can then be used to access fields that are defined on Person, because Employee derives from Person.

Record expressions can be *explicitly* converted from a object to a derived object. Explicit conversion is done by specifying the object script name. For example, the following calculation will get all Employees, but then treat the result as manager:

```
convert([Manager], all([Employee]))
```

That is, this expression can now access fields and relationships that are now only available on managers. If employees are encountered that are not managers, then the conversion will turn them into null.

Converting from datetime to string

If a calculation that has a **datetime** result data type is converted to a **string**, then the date/time value is firstly transformed into the time zone of the user who is currently viewing the calculation result (or running the workflow), and the date/time is then formatted using Australian date format.

Time zone adjustments are not, however, made for **date**-only or **time**-only data types.

Implicit conversion and comparisons

If two different data types are compared (for example to check whether one is larger than the other) then implicit conversion can contribute to accidental calculation errors. For example, if a **number** were to be accidentally compared to a **string** then this could cause unintended behavior. So help prevent this type of error, the calculation builder will give warnings and/or errors for suspicious conversions - including some scenarios where implicit conversion would ordinarily otherwise apply.