eServices Manager Plug-in for GAX

Using Formulas in Field Codes

12/18/2025

Contents

Using Formulas in Field Codes

In addition to system variables such as `Contact.FirstName`, field codes may contain formulas. This section provides an outline of formula usage. Details on many of these topics are provided in the Genesys eServices Field Codes Reference Manual.

You must always delimit field codes by using `<$ … $>`. If you type a field code directly into the body of a standard response, then you must enter the delimiters yourself. If you select from the list of field codes in eServices Manager, then the delimiters are added automatically.

The text that appears inside the delimiters is a formula. Field code formulas are very similar to formulas in other applications, such as Microsoft Excel.

A *formula* is a sequence of one or more operands (such as numbers and text strings), separated by operators (such as + and -).

For example, in the following formula, 2 and 3 are operands and + is an operator:

```
<$2 + 3$>
```

Operands can be values that do not change (constants), or values that vary based on the context. In the previous formula, all the operands are constants, so the formula always evaluates to `5`. The next formula, on the other hand, evaluates to a different value for each agent who uses it:

```
<$Agent.Signature$>
```

Field Code Syntax

To summarize field code syntax:

- A field code must be delimited by `<$ ... $>`.
- Alphabetic strings, whether constants in formulas or elsewhere in a field code, must be enclosed in double quotes.
- Numeric constants require no special treatment.
- You must use special characters for some purposes. For example, for your field code to render with a line break, you cannot simply type a carriage return. Instead, you must insert the expression `\n`. A list of these special characters is available.

HTML in Field Codes

With special configuration, field codes can contain HTML markup; for example, you could have a field code <$my.agent.signature$> defined as

```
Sam Agent<BR />
Acme Products<BR />
29 Exterior Blvd<BR />
Springfield, CX 09090<BR />
```

To enable this, you must use the Java property `-Dsrl-field-code-allow-html=true`, in one of the following ways:

- Add it to the `JavaArgs` section of `ContactServerDriver.ini`
- Add it as an argument to the startup command line in `contactServer.sh`.

Operator Precedence

If you use more than one operator in a formula, the order in which they are evaluated depends on their relative *precedence* (higher precedence operators are evaluated first). For example, multiplication (`*`) has a higher precedence than addition (`+`), so that the formula below evaluates to `14`, not `20`:

```
<$2 + 3 * 4$>
```

You can use parentheses to override the default precedence. The formula below evaluates to `20`:

```
<$(2 + 3) * 4$>
```

For a complete list of operators and their relative precedence, see "Operator Precedence" in the Genesys eServices Field Codes Reference Manual.

Data Types

Operands of several different types may appear in formulas:

- Number
- String (text)
- Date/time
- Boolean (true/false)
- Object (Contact, Interaction, and Agent)

Each data type behaves differently in formulas, and the operators have different meanings when you use them with different data types. For example, the `+` operator means "add" when used with numbers, but "concatenate" (paste together) when used with strings. This formula evaluates to *Uncle Sam Wants You*

```
<$"Uncle Sam " + "Wants You"$>
```

In addition, some operators cannot be used with some data types at all. For example, you cannot use the multiplication (`*`) operator on two strings.
All formulas, regardless of their final data type, are converted to strings before being merged into your standard response. This conversion follows a set of default rules that depend on the data type. For example, the default rules for numbers round them off to integers. This formula causes 2 to be

inserted into your standard response, even though the real result is `2.25`:

```
<$9 / 4$>
```

You can use the `Text` function (see below) or `format operator`:) to override the default formatting. Either of the following formulas inserts `2.25` into your standard response:

```
<$Text(9 / 4, "#.##")$>
```

```
<$(9 / 4):"#.##"$>
```

For a detailed list of data types and how you can use them, see "Data Types" in the Genesys eServices Field Codes Reference Manual..

Functions

When composing formulas, you can use many built-in functions. *Functions* are predefined formulas that perform calculations using values, called *arguments,* which you supply. To use a function, write its name, followed by an opening parenthesis, the arguments for the function separated by commas, and a closing parenthesis.

Function arguments may be of any data type, although individual functions may place restrictions on their arguments. Function arguments may be constants or formulas. The `Length` function, for example, takes a single string argument and returns its length in characters. This formula evaluates to `13`:

```
<$Length("Hello, world!")$>
```

As another example, the `Date` function takes individual date components (year, month, day, and so on), and constructs a date/time value. The formula below evaluates to `2019-11-23 09:03:10: 2019`

```
<$Date(2019, 11, 23, 9, 3, 10)$>
```

Functions may act as arguments to other functions. The `WeekdayName` function takes a single date/time argument and returns the day of the week as a string. The formula below evaluates to `Tuesday`:

```
<$WeekdayName(Date(2019, 11, 23, 9, 3, 10))$>
```

This formula evaluates to `7`:

```
<$Length(WeekdayName(Date(2019, 11, 23, 9, 3, 10)))$>
```

For detailed descriptions of all available functions, see "Functions" in the Genesys eServices Field Codes Reference Manual.

Important

If you want to combine data types, you must first convert the data types to text. Consider the following example:

```
<$ Agent.FirstName + Interaction.DateCreated $>
```

This formula causes an error, as it mixes two data types: Text (**Agent.FirstName**) and Date (**Interaction.DateCreated**). Instead, use the Text type for both types, as shown below:

```
Agent.FirstName + Text(Interaction.DateCreated)
```

Using Objects

All object/property pairs are also available in the `Variables` drop-down menu in the eServices Manager `Field Code Editor`.
Object properties can be of any data type. `Agent.FullName`, for example, is a string, but `Interaction.DateCreated` is a date/time.
The data type of an object property can even be another object. For example, `Contact.EmailAddresses` yields another object called a `ContactEmailAddressList`. In cases such as this, you can access the properties of the resulting object by entering a period (.), followed by the property name, just as before. For example, the formula below evaluates to the number of email addresses assigned to the contact:

```
<$Contact.EmailAddresses.Count$>
```

Some object properties require arguments just as functions do. For these properties, write the arguments, enclosed in parentheses after the property name, just as before.

For example, the ContactEmailAddressList object has a property named `Exists`, which you can use to test whether a particular email address is assigned to a contact. The data type of this property is Boolean (true/false), and it takes one argument, the email address to test. For example:

```
<$Contact.EmailAddresses.Exists("samd@acme.com")$>
```

For detailed descriptions of all objects and their properties, see "Objects" in the Genesys eServices Field Codes Reference Manual.