How-to: Interaction via HTTP protocol

Example 1

Task

We have a certain set of cities associated with their countries.

```
CLASS Country 'Country';
id 'Code' = DATA STRING[20]
(Country) IN id;
name 'Name' = DATA ISTRING[100
] (Country) IN id;

country (STRING[20] id) = GRO
UP AGGR Country c BY id(c);

CLASS City 'City';
name 'Name' = DATA ISTRING[100
] (City) IN id;

country 'Country' = DATA
Country (City);
nameCountry 'Country' (City
c) = name(country(c));

FORM cities 'Cities'
    OBJECTS c = City
    PROPERTIES(c) name,
nameCountry, NEW, DELETE
;

NAVIGATOR {
    NEW cities;
}
```

We need to send an HTTP request for adding a city in the JSON format to a certain url.

Solution

```
postCity 'Send' (City c)  {
    EXPORT JSON FROM countryId = id(country(c)), name
= name(c);

    LOCAL result = FILE();
    EXTERNAL HTTP 'http://localhost:7651/exec?
action=Location.createCity' PARAMS exportFile() TO
result;

    LOCAL code = STRING[10]();
    LOCAL message = STRING[100]();
    IMPORT JSON FROM result() TO() code, message;
    IF NOT code() == '0' THEN {
        MESSAGE 'Error: ' + message();
    }
}

EXTEND FORM cities
    PROPERTIES(c) postCity
;
```

The EXPORT operator will create a JSON in the FILE format and then will write it to the exportFile property. Here is an example of the generated file:

```
{"countryId":"123","name":"San Francisco"}
```

Then we call the EXTERNAL operator, which sends a request to the specified url passing there the contents of the generated file as Body. In this case, since the property in the FROM block has the type JSON, *application/json* will be used as the content type. <namespace><property name> is encoded in the url. In this case, the namespace of the action being called (**createCity**) is **Location**. All parameters are passed consequently with the ID **p**. The response from the server will be written to the **result** property. Suppose that the response is received in the JSON format and has one of the following types:

```
{"code":"0","message":"OK"}
```

```
{"code":"1","message":"Invalid country code"}
```

The response is handled by the IMPORT operator which parses the corresponding parameters into the **code** and **message** properties respectively. If any error occurs, the user will see a corresponding error message.

Example 2

Task

Similar to **Example 1**.

We need to handle the incoming HTTP request and create a new city in the database with the parameters provided in the request.

Solution

```
createCity (FILE f)  {

    LOCAL cy = STRING[20] ();
    LOCAL ne = STRING[100] ();

    IMPORT JSON FROM f AS FILE TO()
cy = countryId, ne = name;

    IF NOT country(cy()) THEN {
        EXPORT JSON FROM code = '1',
message = 'Invalid country code';
        RETURN;
    }

    NEW c = City {
        name(c) <- ne();
        country(c) <- country(cy());

        APPLY;
    }

    EXPORT JSON FROM code = '0',
message = 'OK';
}
```

Since the property is named **createCity** and located in the module with the namespace **Location**, the url on which the request will be handled looks like this:

```
http://localhost:7651/exec?action=Location.createCity
```

Body of the HTTP request will be passed as a parameter of the type **FILE**. The values read from the **countryId** and **name** parameters are written to the local properties **cy** and **ne** respectively.

If there is no country with the corresponding code, then a JSON file is generated similar to that described in the previous example, and the RETURN operator is called to abort execution. By default, the response message value is also stored in the **exportFile** property.

If all the actions are completed successfully, the corresponding "OK message" is generated in response.

Example 3

Task

We have the logic of book orders.

```
CLASS Book 'Book';
id 'Code' = DATA STRING[10]
(Book) IN id;
name 'Name' = DATA ISTRING[100]
(Book) IN id;

book (STRING[10] id) = GROUP AGGR
 Book b BY id(b);

CLASS Order 'Order';
date 'Date' = DATA DATE (Order);
number 'Number' = DATA STRING[10]
(Order);

CLASS OrderDetail 'Order line';
order 'Order' = DATA Order
(OrderDetail) NONULL DELETE;

book 'Book' = DATA Book
(OrderDetail) NONULL;
nameBook 'Book' (OrderDetail d)
= name(book(d));

quantity 'Quantity' = DATA INTEGER
 (OrderDetail);
price 'Price' = DATA NUMERIC[14,2
] (OrderDetail);

FORM order 'Order'
    OBJECTS o = Order PANEL
    PROPERTIES(o) date, number

    OBJECTS d = OrderDetail
    PROPERTIES(d) nameBook,
quantity, price, NEW, DELETE
    FILTERS order(d) == o

    EDIT Order OBJECT o
;

FORM orders 'Orders'
    OBJECTS i = Order
    PROPERTIES(i) READONLY date,
number
    PROPERTIES(i) NEWSESSION NEW,
EDIT, DELETE
;

NAVIGATOR {
    NEW orders;
}
```

We need to send an HTTP request for creating a new order in the JSON format to a certain url.

Solution

```
FORM exportOrder
    OBJECTS order = Order PANEL
    PROPERTIES dt = date(order), nm = number(order)

    OBJECTS detail = OrderDetail
    PROPERTIES id = id(book(detail)), qn = quantity(detail), pr
= price(detail)
    FILTERS order(detail) == order
;

exportOrder 'Send' (Order o)  {
    EXPORT exportOrder OBJECTS order = o JSON;

    LOCAL result = FILE();
    EXTERNAL HTTP 'http://localhost:7651/exec?action=Location.
importOrder' PARAMS exportFile() TO result;
}

EXTEND FORM orders
    PROPERTIES(i) exportOrder;
;
```

To create a JSON with nested tags, we need to create a form with the corresponding objects linked via the **FILTERS** block of operators. Based on the dependencies between objects, the system will generate a JSON file with the corresponding structure. In the considering example, the output JSON structure will look like this:

```
{
   "dt":"20.08.18",
   "nm":"1",
   "detail":[
      {
         "pr":5.99,
         "id":"b1",
         "qn":3
      },
      {
         "pr":6.99,
         "id":"b2",
         "qn":2
      }
   ]
}
```

We do not create a custom tag for "order", since the object value is passed as an argument to the **EXPORT** operator.
In this example, the response to the HTTP request is ignored.

Example 4

Task

Similar to **Example 3**.

We need to handle the incoming HTTP request and create a new order in the database with the parameters provided in the request.

Solution

```
date = DATA LOCAL DATE();
number = DATA LOCAL STRING[10]();

id = DATA LOCAL STRING[10] (INTEGER);
quantity = DATA LOCAL INTEGER (INTEGER);
price = DATA LOCAL NUMERIC[14,2] (INTEGER);
FORM importOrder
    PROPERTIES dt = date(), nm = number()

    OBJECTS detail = INTEGER
    PROPERTIES id = id(detail), qn = quantity(detail), pr
= price(detail)
;

importOrder (FILE f)  {
    IMPORT importOrder JSON FROM f;

    NEW o = Order {
        date(o) <- date();
        number(o) <- number();
        FOR id(INTEGER detail) DO NEW d = OrderDetail {
            order(d) <- o;
            book(d) <- book(id(detail));
            quantity(d) <- quantity(detail);
            price(d) <- price(detail);
        }

        APPLY;
    }
}
```

To import the corresponding file in the JSON format, we need to create a form of a similar structure, except that the INTEGER type will be used as object classes. During the import process, the tag values will be placed in the properties with the corresponding names. The **date** and **number** properties have no parameters, since their values in JSON are provided at the topmost level.

Example 5

Condition

Similar to **Example 4**.

We need to send an HTTP request to create an order in the JSON format to a certain url as in the previous example, except that everything must be wrapped in the **order** tag.

Solution

```
GROUP order;
FORM exportOrderNew
    OBJECTS o = Order
    PROPERTIES IN order dt = date(o), nm = number(o)

    OBJECTS detail = OrderDetail IN order
    PROPERTIES id = id(book(detail)), qn = quantity(detail), pr
= price(detail)
    FILTERS order(detail) == o
;

exportOrderNew 'Send (new)' (Order o)  {
    EXPORT exportOrderNew OBJECTS o = o JSON;

    LOCAL result = FILE();
    EXTERNAL HTTP 'http://localhost:7651/exec?action=Location.
importOrderNew' PARAMS exportFile() TO result;
}

EXTEND FORM orders
    PROPERTIES(i) exportOrderNew;
;
```

Unlike the previous example, here we create a property group named **order** using the GROUP operator. When declaring a form, we put all the properties of the purchase order as well as the "detail" object into this property group. The result JSON will look like this:

```
{
   "order":{
      "dt":"20.08.18",
      "nm":"1",
      "detail":[
         {
            "pr":5.99,
            "id":"b1",
            "qn":3
         },
         {
            "pr":6.99,
            "id":"b2",
            "qn":2
         }
      ]
   }
}
```

Example 6

Condition

Similar to **Example 5**.

We need to handle the incoming HTTP request and create a new order in the database with the parameters provided in the request.

Solution

```
FORM importOrderNew
    PROPERTIES IN order dt = date(), nm = number()

    OBJECTS detail = INTEGER IN order
    PROPERTIES id = id(detail), qn = quantity(detail), pr
= price(detail)
;

importOrderNew (FILE f)  {
    IMPORT importOrderNew JSON FROM f;

    NEW o = Order {
        date(o) <- date();
        number(o) <- number();
        FOR id(INTEGER detail) DO NEW d = OrderDetail {
            order(d) <- o;
            book(d) <- book(id(detail));
            quantity(d) <- quantity(detail);
            price(d) <- price(detail);
        }

        APPLY;
    }
}
```

Just as in the export process, we put all the properties and the **detail** object to the "order" group to correctly receive the new version of JSON.

Example 7

Task

Similar to **Example 3**.

We need to return a list of order numbers for a given date using an HTTP GET request in which this date is provided.

Solution

```
FORM exportOrders
    OBJECTS date =
DATE PANEL

    OBJECTS order
= Order
    PROPERTIES nm
= number(order)
    FILTERS date
(order) = date
;

getOrdersByDate (DA
TE d) {
    EXPORT
exportOrders OBJECTS
 date = d JSON;
}
```

The url to which the HTTP request should be sent will look like this: http://localhost:7651/exec?action=Location.getOrdersByDate&p=12.11.2018 .

The response JSON will look like this:

```
{
    "order": [
        {
            "nm": "42"
        },
        {
            "nm": "65"
        }
    ]
}
```