Execute Stored Procedure

This plug-in allows a stored procedure to be executed while mapping data in and out. It can work with both complex and simple data types.

Function

This function may be used anywhere expressions are valid to execute a read-only stored procedure. This function **must not** be used to execute stored procedures that modify data and actively attempts to prevent such use. Use the Smart Service to modify data instead.

Inputs

The function with three inputs:

1. **Data Source Name:** The name of the data source to use as specified in custom.properties (conf.data.datasources) and shown in the Query Database Smart Service.

2. **Procedure Name:** The name of the stored procedure to use. For Oracle this may need to be put in capitals. If the procedure belongs to a package, the schema must be present with the package (e.g. `APPIAN.BOOKPKG.SP_LISTBOOKS`)
3. **Procedure Inputs:** The list of IN parameter names and their values.
    i. The name of the ACP must match the name of the stored procedures parameter.
    ii. All IN parameters of the stored procedure must be defined.
4. **Timeout:** The number of seconds to allow the query to execute before timing out and requested to be cancelled. Defaults to 30 seconds. Maximum 300 seconds.

Return Value

The result of the function is a complex object with the following structure:

- **success:** Returns true if the procedure was executed successfully, otherwise false.
- **error:** If execution was not successful this will contain the failure message
- **parameters:** The OUT and INOUT parameters and values.

- **result:** An array of result sets returned by the stored procedure.

Example

The following example shows how to use the executestoredprocedure function using the example stored procedure in Appendix A.

```
=with(
  local!spResult: fn!executestoredprocedure(
    "jdbc/AppianAnywhere",
    "sp_ListBooks",
    {
      { name: "title_search", value: "%pig%" }
    }
  ),
  if(local!spResult.success,
    local!spResult.result,
    local!spResult.error
  )
)
```

Smart Service

The following outlines how to configure the Execute Stored Procedure Smart Service using the example stored procedure in Appendix A.

Inputs

The Smart Service starts with two inputs:

1. **Data Source Name:** The name of the data source to use as specified in custom.properties (conf.data.datasources) and shown in the Query Database Smart Service.

2. **Procedure Name:** The name of the stored procedure to use. For Oracle this may need to be put in capitals. If following the example use: `SP_LISTBOOKS` If the procedure belongs to a package, the schema must be present with the package (e.g. `APPIAN.BOOKPKG.SP_LISTBOOKS`).

An input (ACP) must be added to the Smart Service for each input parameter of the stored procedure. Output parameters are optional.

- IN parameters can be configured to pass values into the stored procedure by setting a `Value`.
- OUT parameters can be configured to save the value into a process variable by configuring `Save into` to save the result into a process variable.
- INOUT parameters act as both IN and OUT parameters so can have both a `Value` and `Save into` set.
- The name of the ACP must match the name of the stored procedures parameter.
    - All IN parameters of the stored procedure must be defined.
    - OUT parameters are optional and do not need to be defined.
- Result sets are returned as CDTs:
    - Each attribute and type of the CDT should match the stored procedure. Any unmatched attributes will be left blank.
    - The name of the CDT does not matter.
    - The CDT should be marked as multiple. The individual attributes must be non-multiple.
    - Sub-CDTs are not supported.
- Result sets not returned through an OUT parameter can be obtained by naming the ACP resultSet1, resultSet2, etc, depending on which result set is required. MySQL and SQL Server must have result sets returned this way.

Logging & Debugging

Additional logging can be enabled by adding the following to `appian_log4j.properties`:

- `log4j.logger.com.appiancorp.ps.ss=DEBUG`

Appendix A

MySQL

Stored Procedure for Function

```
CREATE TABLE book (
        title varchar(50),
        author varchar(20),
        quantity int,
        price float
);
```

```
INSERT INTO book VALUES ('Peppa Pig: Fun at the Fair', 'Collectif', 23, 3.74);
INSERT INTO book VALUES ('Panda Goes to the Olympics', 'Judith Simanovsky', 7, 4.99);
INSERT INTO book VALUES ('Topsy and Tim Visit London', 'Jean Adamson', 15, 3.29);

DELIMITER $$
        CREATE PROCEDURE sp_ListBooks (
                IN title_search varchar(50)
        )
        BEGIN
                SELECT title, author, quantity, price FROM book WHERE title LIKE
title_search;
        END$$
DELIMITER ;
```

Stored Procedure for Smart Service

```
CREATE TABLE book (
        title varchar(50),
        author varchar(20),
        quantity int,
        price float
);

INSERT INTO book VALUES ('Peppa Pig: Fun at the Fair', 'Collectif', 23, 3.74);
INSERT INTO book VALUES ('Panda Goes to the Olympics', 'Judith Simanovsky', 7, 4.99);
INSERT INTO book VALUES ('Topsy and Tim Visit London', 'Jean Adamson', 15, 3.29);

DELIMITER $$
        CREATE PROCEDURE sp_ListBooks (
                IN title_search varchar(50),
                OUT total_cost float
        )
        BEGIN
                SELECT title, author, quantity, price FROM book WHERE title LIKE
title_search;

                SELECT SUM(quantity * price) INTO total_cost FROM book WHERE title
LIKE title_search;
        END$$
DELIMITER ;
```

SQL Server

Stored Procedure for Function

```
CREATE TABLE book (
        title varchar(50),
        author varchar(20),
        quantity int,
        price float
);
```

```
INSERT INTO book VALUES ('Peppa Pig: Fun at the Fair', 'Collectif', 23, 3.74);
INSERT INTO book VALUES ('Panda Goes to the Olympics', 'Judith Simanovsky', 7, 4.99);
INSERT INTO book VALUES ('Topsy and Tim Visit London', 'Jean Adamson', 15, 3.29);

CREATE PROCEDURE sp_ListBooks (
        @title_search varchar(50)
)
AS
BEGIN
        SELECT title, author, quantity, price FROM book WHERE title LIKE
@title_search;
END;
```

Stored Procedure for Smart Service

```
CREATE TABLE book (
        title varchar(50),
        author varchar(20),
        quantity int,
        price float
);

INSERT INTO book VALUES ('Peppa Pig: Fun at the Fair', 'Collectif', 23, 3.74);
INSERT INTO book VALUES ('Panda Goes to the Olympics', 'Judith Simanovsky', 7, 4.99);
INSERT INTO book VALUES ('Topsy and Tim Visit London', 'Jean Adamson', 15, 3.29);

CREATE PROCEDURE sp_ListBooks (
        @title_search varchar(50),
        @total_cost float OUTPUT
)
AS
BEGIN
        SELECT title, author, quantity, price FROM book WHERE title LIKE
@title_search;

        SET @total_cost = (SELECT SUM(quantity * price) FROM book WHERE title LIKE
@title_search);
END;
```

Oracle

Stored Procedure for Function

```
CREATE TABLE "book" (
        "title" varchar2(50),
        "author" varchar2(20),
        "quantity" int,
        "price" float
);

INSERT INTO "book" VALUES ('Peppa Pig: Fun at the Fair', 'Collectif', 23, 3.74);
```

```
INSERT INTO "book" VALUES ('Panda Goes to the Olympics', 'Judith Simanovsky', 7,
4.99);
INSERT INTO "book" VALUES ('Topsy and Tim Visit London', 'Jean Adamson', 15, 3.29);

CREATE OR REPLACE PROCEDURE sp_ListBooks (
        title_search IN varchar2,
        books_cursor OUT SYS_REFCURSOR
)
AS
BEGIN
        OPEN books_cursor FOR SELECT "title", "author", "quantity", "price" FROM
"book" WHERE "title" LIKE title_search;
END;
```

Stored Procedure for Smart Service

```
CREATE TABLE "book" (
        "title" varchar2(50),
        "author" varchar2(20),
        "quantity" int,
        "price" float
);

INSERT INTO "book" VALUES ('Peppa Pig: Fun at the Fair', 'Collectif', 23, 3.74);
INSERT INTO "book" VALUES ('Panda Goes to the Olympics', 'Judith Simanovsky', 7,
4.99);
INSERT INTO "book" VALUES ('Topsy and Tim Visit London', 'Jean Adamson', 15, 3.29);

CREATE OR REPLACE PROCEDURE sp_ListBooks (
        title_search IN varchar2,
        total_cost OUT float,
        books_cursor OUT SYS_REFCURSOR
)
AS
BEGIN
        OPEN books_cursor FOR SELECT "title", "author", "quantity", "price" FROM
"book" WHERE "title" LIKE title_search;

        SELECT SUM("quantity" * "price") INTO total_cost FROM "book" WHERE "title"
LIKE title_search;
END;
```