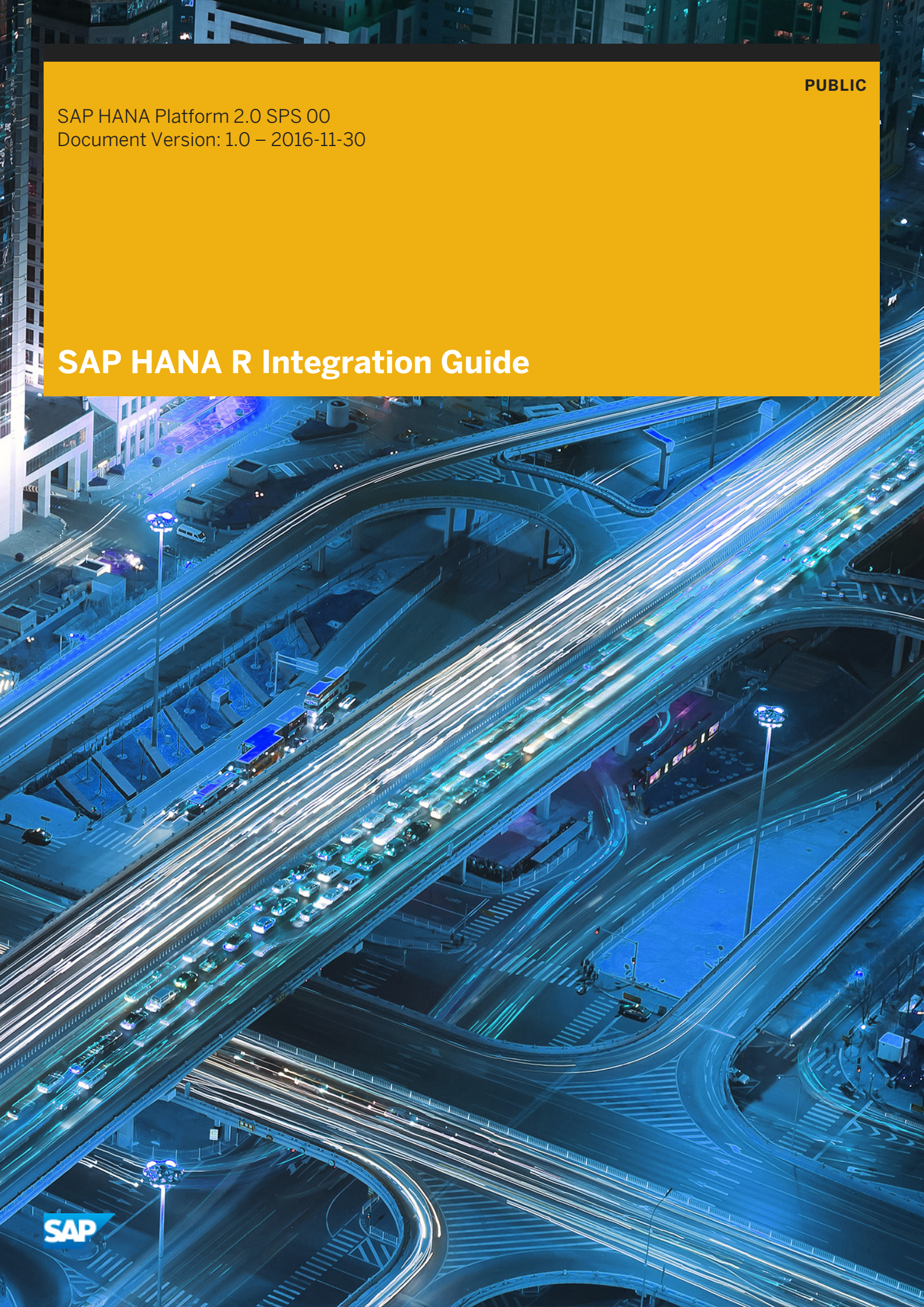
PUBLIC

SAP HANA Platform 2.0 SPS 00
Document Version: 1.0 – 2016-11-30

SAP HANA R Integration Guide



Content

1 What is R?. 3

2 R and SAP HANA. 4

2.1 Supported Data Structures and Data Types. 5

2.2 Security for R. 6

2.3 Best Practices. 7

R Integration with SAP HANA. 7

R Programming. 7

3 Installing and Configuring R. 9

3.1 Install R. 9

3.2 Install and Configure Rserve. 11

3.3 Configure SAP HANA. 12

3.4 Set Up Rserve Authentication. 13

3.5 Set Up SSL/TLS from SAP HANA to Rserve. 15

4 Using R with SAP HANA. 17

4.1 Install the Kernlab Package. 17

4.2 Preparing R Data. 18

4.3 Calling an R Function. 19

4.4 Calling an R Procedure from a SQLScript Procedure. 19

4.5 Saving and Reusing an R Model. 20

4.6 Integrate a Custom R Script Node in Application Function Modeler 22

5 Debugging and Tracing. 24

5.1 Debugging R Procedures by Tracing R Script Output. 24

Trace R Procedure Output Automatically. 24

Trace R Procedure Output Manually. 27

5.2 Trace Detailed Execution Time of R Procedures. 29

5.3 Measuring Performance in R. 30

5.4 Profiling R Procedures with Rprof. 30

1 What is R?

R is an open source programming language and software environment for statistical computing and graphics. The R language has become very popular among statisticians and data miners for developing statistical software and is widely used for advanced data analysis.

The goal of the integration of the SAP HANA database with R is to enable the embedding of R code in the SAP HANA database context. That is, the SAP HANA database allows R code to be processed in-line as part of the overall query execution plan. This scenario is suitable when an SAP HANA-based modeling and consumption application wants to use the R environment for specific statistical functions.

An efficient data exchange mechanism supports the transfer of intermediate database tables directly into the vector-oriented data structures of R. This offers a performance advantage compared to standard SQL interfaces, which are tuple based and, therefore, require an additional data copy on the R side.

SAP does not ship the R environment with the SAP HANA database, as R is open source and is available under the General Public License. SAP does not provide support for R. In order to use the SAP HANA integration with R, you need to download R from the open-source community and configure it, as described in the following chapters. You also need Rserve, a TCP/IP server that allows other programs to use facilities of R without the need to initialize R or link with the R library.

Related Information

http://www.r-project.org/

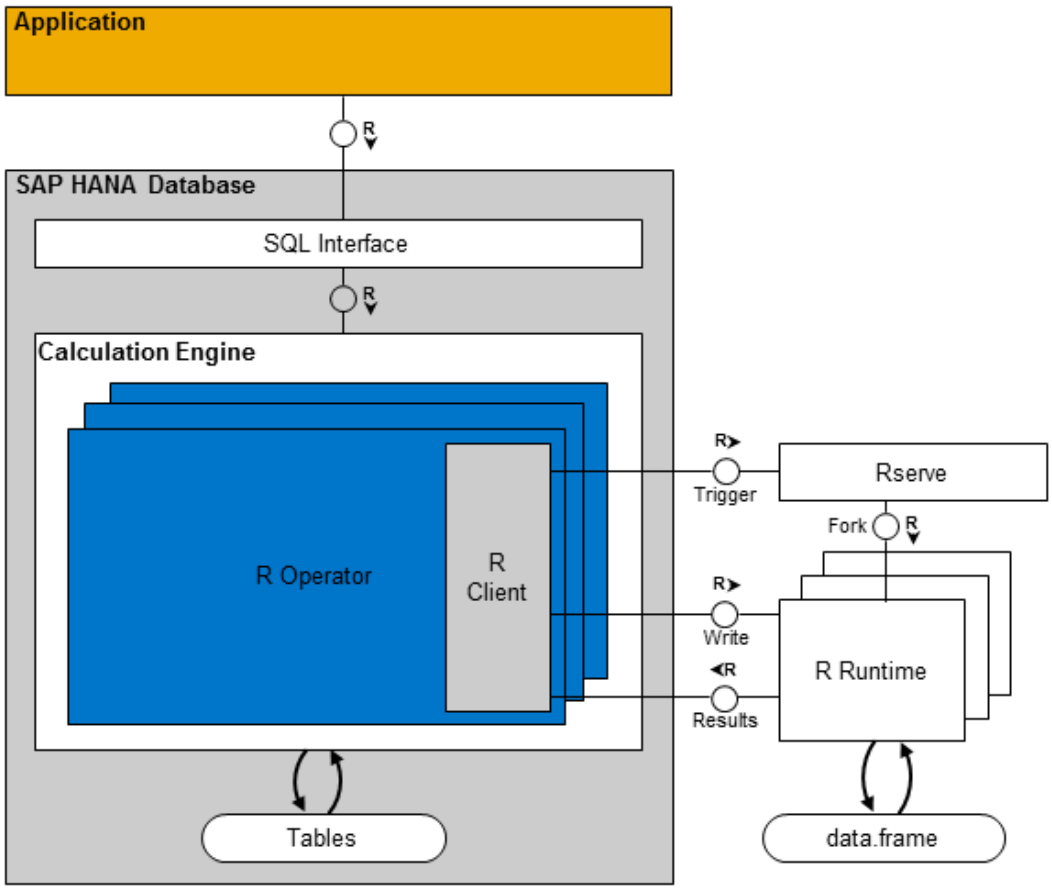
http://www.rforge.net/Rserve/index.html

2 R and SAP HANA

To process R code in the context of the SAP HANA database, the R code is embedded in SAP HANA SQL code in the form of a RLANG procedure. The SAP HANA database uses the external R environment to execute this R code, similarly to native database operations like joins or aggregations. This allows the application developer to elegantly embed R function definitions and calls within SQLScript and submit the entire code as part of a query to the database.

To achieve this, the calculation engine of the SAP HANA database was extended. The calculation engine supports data flow graphs (calculation models) describing logical database execution plans. A node in this data flow graph can be any native database operation, but also a custom operation. One of those custom operations is the R operator.

Like any other operator of the calculation model, the R operator consumes a number of input objects (e.g., intermediate tables retrieved from previously computed operations or other data sources like a column or row store table) and returns a result table. In contrast to native database operations, custom operators are not restricted to a static implementation and can be adjusted for each node independently. In the case of the R operator, this is accomplished by the R function code, which is passed as a string argument to the operator.



The figure above shows three main components of the integrated solution: the SAP HANA-based application, the SAP HANA database, and the R environment.

When the calculation model plan execution reaches an R-operator, the calculation engine's R-client issues a request through the Rserve mechanism to create a dedicated R process on the R host. Then, the R-Client efficiently transfers the R function code and its input tables to this R process, and triggers R execution. Once the R process completes the function execution, the resulting R data frame is returned to the calculation engine, which converts it. Since the internal column-oriented data structure used within the SAP HANA database for intermediate results is very similar to the vector-oriented R data frame, this conversion is very efficient.

A key benefit of having the overall control flow situated on the database side is that the database execution plans are inherently parallel and, therefore, multiple R processes can be triggered to run in parallel without having to worry about parallel execution within a single R process.

Current Limitations

- The R Integration has only been tested by SAP with the R environment installed on SLES 11.
- Only table types are supported as parameters in SQLScript procedures of language RLANG.
  - If you need to pass scalar parameters, they have to be passed as tables/data frames to R.
  - If you need to transfer lists, matrixes or other R data structures from R to SAP HANA, they have to be converted to data frames using the `as.data.frame()` function.
- The variable names from the procedure definition should not contain uppercase letters. Therefore, the variable names in R should also not contain upper-case letters.
- Embedded R functions must have at least one result, in the form of a data frame.
- Factor columns can only be retrieved as character vectors from SAP HANA. Some R functions may require string columns of data frames to be factor, so you may need to convert the input with `as.factor()` before usage.

2.1 Supported Data Structures and Data Types

The main data structure supported to exchange data between the SAP HANA database and the R environment is the R data frame, which has a similar data structure to a column table in the SAP HANA database. The supported data types are listed below:

Table 1: Supported Data Types in R

R Type	SAP HANA SQL Type
numeric (integer)	TINYINT
	SMALLINT
	INTEGER
numeric (double)	REAL

R Type	SAP HANA SQL Type
	DOUBLE
	FLOAT
	FLOAT(p)
	DECIMAL
	DECIMAL(p,s)
	BIGINT
character / factor	VARCHAR
	NVARCHAR
	CLOB
	NCLOB
Date	DATE
DateTime(POSIXct)	TIMESTAMP
	SECONDDATE
Raw	VARBINARY
	BLOB

> **Note**
>
> The datatypes `FLOAT(p)`, `DECIMAL`, `BIGINT` and `DECIMAL(p,s)` might have a higher precision/scale in SAP HANA than in R. Therefore, you might lose accuracy.
>
> The data exchange of character columns with the R environment should be minimized because the transfer of characters can be time-consuming. It is recommended to consider if it is necessary for a given character column to be transferred to the R environment or whether it can be substituted with integer or double data type.
>
> In the case of the R data type factor, the transfer from R to the SAP HANA database is supported, but the factor vector is treated and stored as character column.

2.2 Security for R

R procedures can contain code that can harm security on the server where the Rserve is running, such as the following:

- Access file system (read/write).
- Install new add-on/R packages that can contain binary code (for example, written in C).
- Execute operating system commands.
- Open network connections and download files or open connections to other servers.

SAP HANA comes with a special system privilege that is required to run R procedures, and you should grant the *CREATE R SCRIPT* privilege only to trusted database users who are allowed to create R procedures.

To grant the privilege, a user who already has this privilege (such as the *SYSTEM* user), along with ability to grant it to others, can execute the following SQL command:

```
GRANT CREATE R SCRIPT TO <user> [WITH ADMIN OPTION]
```

Change `<user>` to the name of the user you want to grant the permission.

Rserve Authentication

To enhance security, you should require authentication for calls from SAP HANA to the Rserve. The Rserve enables you to specify a list of users who are allowed to run R code, and SAP HANA enables you to send credentials for all calls to the Rserve. You can also specify that passwords be encrypted during authentication, which is also recommended.

Related Information

Set Up Rserve Authentication [page 13]
SAP HANA Security Guide

2.3 Best Practices

2.3.1 R Integration with SAP HANA

- Avoid unnecessary data transfer between SAP HANA and R.
  - Transfer only the columns and records needed for the calculation on the R side.
  - If possible, avoid transferring character columns.
- For parallelization, try to split your logic into multiple R procedures and execute these using the capabilities provided by SQLScript and the calculation engine.
- Always reference columns by name instead of by index. If you need an index, use the `which()` function to retrieve the index from the name.

2.3.2 R Programming

- Input to R only required data.
- Predefine right-sized data structures and fill them, rather than appending new elements (which involves more expensive copy operations).
- Vectorize calculations, avoid iterations.
  - Use `apply()` and the `plyr` package instead of loops.
  - See http://nsaunders.wordpress.com/2010/08/20/a-brief-introduction-to-apply-in-r .
  - See http://www.r-project.org/doc/Rnews/Rnews_2008-1.pdf .
- Avoid unnecessary character creation operations, for example:

```
USE.NAMES=FALSE in sapply, use.names=FALSE in unlist
```

- Use appropriate functions, often from specialized packages.
- Identify appropriate algorithms, for example, the intrinsic `%in%` operation is O(N), whereas coding this operation in R code might have a cost of $O(N^2)$.

3 Installing and Configuring R

The R and Rserve environments have to be installed on a separate host. You cannot install R on the SAP HANA host. This guide assumes an R installation on a Linux system, preferably SLES Linux; no other R hosting environments are currently supported. The R/Rserve host has to be reachable from the SAP HANA host.

The process consists of three steps:

1. Install R (on a separate host).
2. Install Rserve (on a separate host).
3. Configure SAP HANA.

For high availability, it is possible to install R/Rserve on multiple distinct hosts. When one of them becomes unreachable, another host can take over.

3.1 Install R

Use this procedure to install R and Rserve for use with SAP HANA.

Prerequisites

Make sure the following Linux packages are installed. Use the newest packages available for your Linux version.

- `xorg-x11-devel`
- `gcc-fortran`
- `readline-devel` (install only if you want to use R as a standalone application)
- `libgfortran46` (install only if you use SLES 11 SP2)

Context

To install R for SAP HANA, you must compile the R package from its source code. SAP has tested the SAP HANA integration with R version 2.15 only.

Execute the following steps as user root:

Procedure

1. To compile R, download the R source package from the R Project for Statistical Computing website.

   For information about supported versions of R and Rserve, see 2185029.

2. Extract it to a user defined directory. Go into this directory, and execute:

   ```
   ./configure --enable-R-shlib
   ```

   If you have trouble during configuration, try the following options:

   ```
   --with-readline=no
   --with-x=no
   ```

   If you want to have terminal/history support in R, install the Linux package `readline-devel`.

3. Compile R with the following commands:

   ```
   make clean
   make
   make install
   ```

Results

After compilation, the `R` command is installed in `/usr/local/bin`. If you decide to install it into another directory, make sure that it is properly set in your `PATH` variable.

Related Information

http://cran.r-project.org/doc/manuals/R-admin.html
http://www.r-project.org/

3.2 Install and Configure Rserve

Use this procedure to install and configure Rserve for use with SAP HANA.

Prerequisites

Ensure that you install Rserve on the same host as R.

Context

SAP has tested the integration with Rserve. For information about supported versions of R and Rserve, see 2185029.

Perform the following steps as root user. The user with which you want to use R is called **ruser**.

Procedure

1. Install the Rserve package.
   a. Download the Rserve package fromRserve: Binary R server or from R Project for Statistical Computing .
   b. Login as root user and install the package using the following R terminal command:

```
R
install.packages("/PATH/TO/YOUR/Rserve.tar.gz", repos = NULL)
library("Rserve") # test if installation worked, it should return no output
q()
```

2. Create a non-privileged user called **ruser**.
3. Configure Rserve. As root user, create the file `/etc/Rserv.conf` with the following content and grant file read-access rights to the **ruser** user:

```
maxinbuf 10000000
maxsendbuf 0
remote enable
```

   The value 10000000 is merely an example. We recommend that you set the value of maxinbuf to (physical memory size, in bytes) / 2048. For example if you installed R on a host with 256 GB of physical memory you should set maxinbuf to 134217728.
4. Start Rserve, login as **ruser**, and enter the following:

```
R CMD Rserve --RS-port <PORT> --no-save --RS-encoding utf8
```

   The port for starting Rserve has to be chosen according to the `cer_rserve_port` value in the `indexserver.ini` file, as described in Configure SAP HANA [page 12].

`<PORT>` is the port number (e.g., 30120). The `--no-save` option makes sure that the invoked R runtimes do not store the R environment onto the file system after the R execution has been stopped. This is important to avoid the file system to be filled over time due to multiple R runs.

There is currently no support for automatically starting the Rserve server after rebooting the Linux host. To accomplish this, you can use `crontab` using a shell script like the following, which starts a new Rserve process if none is running:

```
pgrep -u ruser -f "Rserve --RS-port <PORT> --no-save --RS-encoding utf8" || R
CMD Rserve --RS-port <PORT> --no-save --RS-encoding utf8
```

3.3 Configure SAP HANA

Prerequisites

Depending on your system landscape and your R requirements, you may need to modify some of the SAP HANA database configurations, from the *Administration* editor in the SAP HANA studio.

- To enable calling R procedures from the index server, the configuration parameters are under *indexserver.ini* > *calcEngine*.
- To enable calling R procedures from SAP HANA XS, for example, from server-side JavaScript code, the configuration parameters are under *xsengine.ini* > *calcEngine*.

Procedure

1. In the SAP HANA studio, right-click your system in the *SAP HANA Systems* view, and select *Administration*.
2. Select the *Configuration* tab.
3. Select either *indexserver.ini* > *calcEngine* or *xsengine.ini* > *calcEngine*. If the *calcEngine* section is missing, add it.
4. You can add the following parameters:

Table 2:

Parameter	Description	Default
`cer_rserve_addresses`	List of locations where Rserve is running. Each location is specified in the following format: `host1:port1,host2:port2`, where the host is given by its IPv4 address. Use multiple hosts to implement high availability.	None
`cer_timeout`	Connection timeout in seconds. This parameter defines the maximum runtime allowed for the execution of a single R procedure. If you expect your R processing to run longer than 5 minutes, modify this parameter; otherwise R processing may be stopped before completion.	300
`cer_rserve_maxsendsize`	Maximum size of a result transferred from R to SAP HANA, in kilobytes. If the result exceeds the limit, the transfer is aborted with an error.	0 (i.e., no limit)

3.4 Set Up Rserve Authentication

To make sure only authorized users and programs are allowed to connect to the Rserve, you can require authentication when connecting with the Rserve.

Context

You configure authentication by creating an Rserve password file with the names of one or more users who are allowed to access Rserve. The file is a list of username/password entries, one user per line; none should be existing operating systems users. In addition, you can choose between requiring plaintext or encrypted passwords; it is recommended to use encrypted passwords, which is the default.

Procedure

1. On the Rserve side, set the following parameters in the `/etc/Rserv.conf` file:
   - `auth required`: Enables authentication. Set to `disable` to disable authentication.
   - `plaintext disable`: Requires encrypted passwords. Set to `enable` to enable plaintext passwords.
   - `pwdfile` <password file>: Specifies the file with a list of authorized users. Set the filename to `none` to disable authentication.

2. On the SAP HANA system, do the following:
   a. In the connection string (`cer_rserve_addresses` parameter), add the authentication mode, either `plain` or `crypted`, as in the following example:

      ```
      host1:port1:crypted
      ```

      If no authentication mode is specified, no authentication is required.
   b. Store the user name and password for connecting to Rserve in the SAP HANA secure internal credential store, by executing the following SQL as a user with the system privilege *CREDENTIAL ADMIN*:

      ```
      CREATE CREDENTIAL FOR COMPONENT 'RINTEGRATION' PURPOSE
      '<host:port[:authenticationMode]>' TYPE 'PASSWORD' USING
      '<user>:<password>'
      ```

      Replace `<user>` and `<password>` with the user and password of the user who is connecting to the Rserve.

Example

You have a running Rserve on the IP address 1.2.3.4 port 9999, and you want to use authentication with encrypted passwords.

In the Rserve, set the following parameters in your `/etc/Rserv.conf` file:

```
auth required
plaintext disable
pwdfile /home/ruser/passwordfile
```

The Rserve file `/home/ruser/passwordfile` contains the following user and password:

```
myRserveUserName myRservePW
```

On SAP HANA, in the *indexserver.ini* > *calcEngine* > *cer_rserve_addresses* configuration parameter, enter the address `1.2.3.4:9999:crypted`. Then, execute the following SQL statement (as a user with system privilege *CREDENTIAL ADMIN*):

```
CREATE CREDENTIAL FOR COMPONENT 'RINTEGRATION' PURPOSE '1.2.3.4:9999:crypted'
TYPE 'PASSWORD' USING 'myRserveUserName:myRservePW'
```

Related Information

SAP HANA Security Guide

3.5 Set Up SSL/TLS from SAP HANA to Rserve

Starting with version 1.7-0, Rserve supports Secure Sockets Layer (SSL) and/or Transport Layer Security (TLS) connections.

Context

SAP recommends that you use an SSL/TLS channel when you need to transmit sensitive data or when you require secure authentication. For more information, see the *SAP HANA Security Guide*. The link to this guide is located in the *Related Information* section.

To set up the encrypted SSL/TLS channel from SAP HANA to Rserve, perform the following steps.

Procedure

1. Using Rserve, generate a private key and self-signed certificate.

   You can use Open SSL directives, such as:

   - `openssl genrsa -out Rserve.key 2048`
   - `openssl req -new -key Rserve.key -out Rserve.csr`
   - `openssl x509 -req -days 365 -in Rserve.csr -signkey Rserve.key -out Rserve.crt`

2. In the `/etc/Rserv.conf` file, set the following parameters.

   - `tls.key <private key PEM file>`
   - `tls.cert <server certificate PEM file>`
   - `tls.port <port for SSL/TLS channel>`
   - `qap disable # to disable the non crypted port`

3. Start Rserve.

   With this action, a server is created to listen to the TLS port.

4. On the SAP HANA system, import the certificate that you generated in step 1 into the secure store.

   Use the following code sample as an example of how to do this.

   ```
   sapgenpse maintain_pk -a <Rserve certificate PEM file> -p $SECUDIR/sapsrv.pse
   ```

   > **Tip**
   >
   > You can use the command `sapgenpse -h` to see more information.

5. In the connection string, `cer_rserve_addresses parameter`, add an @ (at sign) at the end of the string to indicate that this address uses an SSL connection, as shown in the following example.

   ```
   host1:port1@
   ```

You can also use authentication mode, as shown in the following example.

```
host1:port1:crypted@
```

If you use multiple Rserve addresses, add the “@” sign after each Rserve address that requires SSL, as shown in the following example.

```
host1:port1@,host2:port2@
```

4 Using R with SAP HANA

In this section there are examples of calling an R procedure. For these examples to work, you must prepare your data and install the R package kernlab.

> **Note**
>
> You do not have to install kernlab for productive systems. This is only needed to run the examples in this section.

To create and call R procedures, you must have the system privilege `CREATE R SCRIPT`.

4.1 Install the Kernlab Package

In this section, there are examples of calling an R procedure. For these examples to work, you must install the R package kernlab, version 0.9-14.

Context

.

> **Note**
>
> You do not have to install kernlab for productive systems. This is only needed to run the examples in this section.

Procedure

1. Download the kernlab package fromKernel-based Machine Learning Lab or from R Project for Statistical Computing .
2. Log in as user **`ruser`** and install the kernlab package using the following R terminal command:

```
R
install.packages("/PATH/TO/YOUR/kernlab.tar.gz", repos = NULL)
library("kernlab") #test if installation worked, it should return no output
q()
```

4.2 Preparing R Data

To create some initial data for our first example, we use the dataset `spam` provided by R and upload it to the SAP HANA database. For now, you do not need to understand the R procedure.

In the SAP HANA studio, execute the following SQL statements, for example, in the SQL Console for your SAP HANA system.

```
DROP TABLE "spam";
CREATE COLUMN TABLE "spam"(
"make" DOUBLE, "address" DOUBLE, "all" DOUBLE, "num3d" DOUBLE, "our" DOUBLE,
"over" DOUBLE, "remove" DOUBLE, "internet" DOUBLE, "order" DOUBLE, "mail"
DOUBLE, "receive" DOUBLE, "will" DOUBLE, "people" DOUBLE, "report" DOUBLE,
"addresses" DOUBLE, "free" DOUBLE, "business" DOUBLE, "email" DOUBLE, "you"
DOUBLE, "credit" DOUBLE, "your" DOUBLE, "font" DOUBLE, "num000" DOUBLE, "money"
DOUBLE, "hp" DOUBLE, "hpl" DOUBLE, "george" DOUBLE, "num650" DOUBLE, "lab"
DOUBLE,
"labs" DOUBLE, "telnet" DOUBLE, "num857" DOUBLE, "data" DOUBLE, "num415" DOUBLE,
"num85" DOUBLE, "technology" DOUBLE, "num1999" DOUBLE, "parts" DOUBLE,
"pm" DOUBLE, "direct" DOUBLE, "cs" DOUBLE,"meeting" DOUBLE, "original" DOUBLE,
"project" DOUBLE, "re" DOUBLE, "edu" DOUBLE, "table" DOUBLE, "conference"
DOUBLE, "charSemicolon" DOUBLE, "charRoundbracket" DOUBLE, "charSquarebracket"
DOUBLE, "charExclamation" DOUBLE, "charDollar" DOUBLE, "charHash" DOUBLE,
"capitalAve" DOUBLE, "capitalLong" DOUBLE, "capitalTotal" DOUBLE,
"type" VARCHAR(5000), "group" INTEGER);
DROP PROCEDURE LOAD_SPAMDATA;
CREATE PROCEDURE LOAD_SPAMDATA(OUT spam "spam")
LANGUAGE RLANG AS
BEGIN
  ##--if the kernlab package is missing see Requirements
  library(kernlab)
  data(spam)
  ind <- sample(1:dim(spam)[1],2500)
  group <- as.integer(c(1:dim(spam)[1]) %in% ind)
  spam <- cbind(spam, group)
END;
DROP TABLE "spamTraining";
DROP TABLE "spamEval";
CREATE COLUMN TABLE "spamTraining" like "spam";
CREATE COLUMN TABLE "spamEval" like "spam";
DROP PROCEDURE DIVIDE_SPAMDATA;
CREATE PROCEDURE DIVIDE_SPAMDATA()
AS BEGIN
CALL LOAD_SPAMDATA(spam);
Insert into "spamTraining" select * from :spam where "group"=1;
Insert into "spamEval" select * from :spam where "group"=0;
END;
CALL DIVIDE_SPAMDATA();
Alter Table "spamTraining" DROP ("group");
Alter Table "spamEval" DROP ("group");
```

After executing the R procedure, your SAP HANA database instance should contain the following two new tables:

- `spamTraining`
- `spamEval`

4.3 Calling an R Function

This topic demonstrates how the support vector machine classification can be embedded in the SAP HANA database.

In the SAP HANA studio, execute the following SQL statements, for example, in the SQL Console for your SAP HANA system. The code defines and calls an embedded R procedure.

```
DROP TABLE "spamClassified";
CREATE COLUMN TABLE "spamClassified" LIKE "spamEval" WITH NO DATA;
ALTER TABLE "spamClassified" ADD ("classified" VARCHAR(5000));
DROP PROCEDURE USE_SVM;
CREATE PROCEDURE USE_SVM(IN train "spamTraining", IN eval "spamEval",
OUT result "spamClassified")
LANGUAGE RLANG AS
BEGIN
    library(kernlab)
    model <- ksvm(type~. , data=train, kernel=rbfdot(sigma=0.1))
    classified <- predict(model, eval [,-(which(names(eval) %in% "type"))])
    result <- as.data.frame(cbind(eval, classified))
END;
CALL USE_SVM("spamTraining", "spamEval", "spamClassified") WITH OVERVIEW;
SELECT * FROM "spamClassified";
```

The first part creates a table whose schema is derived from the table `spamEval`, and adds a column for the classification result.

The schema is used in the second part of the code to define the output of the function `USE_SVM`. The language `RLANG` is used to indicate that our procedure is an R procedure, expecting R code between `BEGIN` and `END`.

The variables `train`, `eval` and `result` (defined in the `CREATE PROCEDURE` statement) correspond to the variables in the R environment that refer to the data frames. This means that the input tables `spamTraining` and `spamEval` passed to the `USE_SVM` procedure are internally transferred to the R environment and provided there as data frames. Consequently, the variable `result` has to be an R data frame as well for the `USE_SVM` procedure to return results.

4.4 Calling an R Procedure from a SQLScript Procedure

We now demonstrate how an R procedure can be called by another procedure, for example, one written in SQLScript.

In the SAP HANA studio, execute the following SQL statements, for example, in the SQL Console for your SAP HANA system.

```
DROP TABLE "spamClassified";
CREATE COLUMN TABLE "spamClassified" LIKE "spamEval" WITH NO DATA;
ALTER TABLE "spamClassified" ADD ("classified" VARCHAR(5000));
DROP PROCEDURE USE_SVM;
CREATE PROCEDURE USE_SVM( IN train "spamTraining", IN eval "spamEval",
OUT result "spamClassified")
LANGUAGE RLANG AS
BEGIN
    library(kernlab)
```

```
    model <- ksvm(type~. , data=train, kernel=rbfdot(sigma=0.1))
    classified <- predict(model, eval [,-(which(names(eval) %in% "type"))])
    result <- as.data.frame(cbind(eval, classified))
END;
DROP PROCEDURE R_PARTOFMORE;
CREATE PROCEDURE R_PARTOFMORE(OUT result "spamClassified")
LANGUAGE SQLSCRIPT AS
BEGIN
    subset1 = select * from "spamEval" where "capitalLong" > 14;
    subset2 = select * from "spamEval" where "capitalLong" <= 14;
    train = select * from "spamTraining";
    newtrain = CE_UNION_ALL(:subset1, :train);
    CALL USE_SVM(:newtrain, :subset2, result);
END;
CALL R_PARTOFMORE("spamClassified") WITH OVERVIEW;
SELECT * FROM "spamClassified";
```

The first part of the code is a direct copy of our previous example. The only difference is that the call of the R procedure `USE_SVM` is now part of the second procedure `R_PARTOFMORE`. This second procedure rearranges the initial training and evaluation data, so that in the R environment more data is used to train the support vector machine.

> **Note**
>
> Instead of referencing the column named `type` by index, we use the R command `which` to access the column by name. This is because the SAP HANA database does not necessarily keep the column order intact.

4.5 Saving and Reusing an R Model

In previous examples, we had a procedure that was used for training a model and used this model directly within the same execution. With the support of binary columns, it is possible to train a model and store it in the database and use it later.

Training the Model

The following is the same code as in the previous example, but adds code to store the model as binary in the database. To do this, we create a table with a BLOB column for holding the model.

In the SAP HANA studio, execute the following SQL statements, for example, in the SQL Console for your SAP HANA system.

```
DROP TYPE SPAM_MODEL_T;
CREATE TYPE SPAM_MODEL_T AS TABLE (
    ID INTEGER,
    DESCRIPTION VARCHAR(255),
    MODEL BLOB
);
DROP TABLE SPAM_MODEL;
CREATE COLUMN TABLE SPAM_MODEL (
    ID INTEGER,
```

```
    DESCRIPTION VARCHAR(255),
    MODEL BLOB
);
DROP PROCEDURE SAPM_TRAIN_PROC;
CREATE PROCEDURE SPAM_TRAIN_PROC (IN traininput "spamTraining", OUT modelresult
SPAM_MODEL_T)
LANGUAGE RLANG AS
BEGIN
    generateRobjColumn <- function(...){
        result <- as.data.frame(cbind(
            lapply(
                list(...),
                function(x) if (is.null(x)) NULL else serialize(x, NULL)
            )
        ))
        names(result) <- NULL
        names(result[[1]]) <- NULL
        result
    }
    library(kernlab)
    svmModel <- ksvm(type~. , data=traininput, kernel=rbfdot(sigma=0.1))
    modelresult <- data.frame(
        ID=c(1),
        DESCRIPTION=c("SVM Model"),
        MODEL=generateRobjColumn(svmModel)
    )
END;
CALL SPAM_TRAIN_PROC("spamTraining", SPAM_MODEL) WITH OVERVIEW;
select * from SPAM_MODEL;
```

Using the Model

The following uses the model stored in the database. It gets the model as an input to the procedure and re-creates the model out of the BLOB column (unserialize). In this example, it is assumed that the table `SPAM_MODEL` contains the model and only this model.

```
DROP TABLE "spamClassified";
CREATE COLUMN TABLE "spamClassified" LIKE "spamEval" WITH NO DATA;
ALTER TABLE "spamClassified" ADD ("classified" VARCHAR(5000));
DROP PROCEDURE USE_SVM;
CREATE PROCEDURE USE_SVM(IN eval "spamEval", IN modeltbl SPAM_MODEL_T, OUT
result "spamClassified")
LANGUAGE RLANG AS
BEGIN
    library(kernlab)
    svmModel <- unserialize(modeltbl$MODEL[[1]])
    classified <- predict(svmModel, eval [,-(which(names(eval) %in% "type"))])
    result <- as.data.frame(cbind(eval, classified))
END;
CALL USE_SVM("spamEval", SPAM_MODEL, "spamClassified") WITH OVERVIEW;
SELECT * FROM "spamClassified";
```

4.6 Integrate a Custom R Script Node in Application Function Modeler

Integrate a custom R Script node in a flowgraph using the application function modeler.

Prerequisites

You need to have R integration in SAP HANA to use the R integration in flowgraphs.

You must have the `CREATE R SCRIPT` system privilege.

Context

If your SAP HANA system has R integration, the *R Script* node template and the compartment *R Examples* appears in the *Node Palette* of the SAP HANA studio application function modeler.

You can integrate an R script to the flowgraph by creating an R Script node. In the script, you can access the properties of the input and output anchors as follows.

Table 3: Anchor property placeholders in Script nodes.

Name of the *i*-th input	$(node.inputs[].name)
Name of the *i*-th output	$(node.outputs[*i*].name)
Name of the *j*-th attribute of the *i*-th input	$(node.inputs[*i*].attributes[*j*].name)
Name of the *j*-th attribute of the *i*-th output	$(node.outputs[*i*].attributes[*j*].name)

You can address the last input, output, or attribute by choosing `i` or `j` as -1.

Procedure

1. Add the *R Script* node to the flowgraph.
2. Edit the node name, its anchors, and its standard properties.
3. Create your custom R program in the *Script* tab of the *Properties* view.

> **Note**
>
> The nodes from the *R Examples* compartment already contain valid R scripts in the *Script* tab of the *Properties* view.

Related Information

SAP HANA Developer Guide for SAP HANA Studio

5 Debugging and Tracing

There are several methods for debugging and tracing in R with SAP HANA. This section provides information for using these features.

5.1 Debugging R Procedures by Tracing R Script Output

One method to debug R procedures is to print out some traces in R scripts and then have users check the printed result. SAP HANA R integration supports recording R script output from `stdout` and `stderr`. You can configure the SAP HANA to record the R script outputs automatically, which subsequently can be seen in the SAP HANA trace file.

A similar method allows you to tweak the R scripts to record part of the R script output manually, and save it in output variables.

Related Information

Trace R Procedure Output Automatically [page 24]
Trace R Procedure Output Manually [page 27]

5.1.1 Trace R Procedure Output Automatically

Set up an output trace to debug an R procedure automatically.

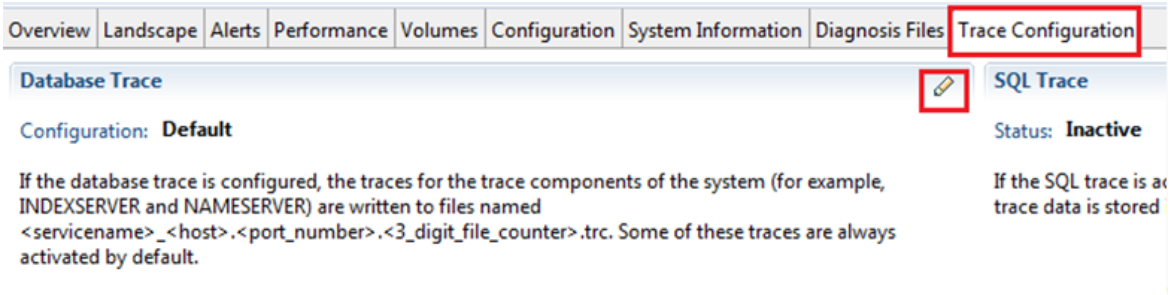
Context

In SAP HANA, you find a trace component called `rscript_output`. If you want to record the R script output, then set the trace level for this component either to `info` or to `debug`. You can see the output in the latest `indexserver_*.trc` file when it is finished.

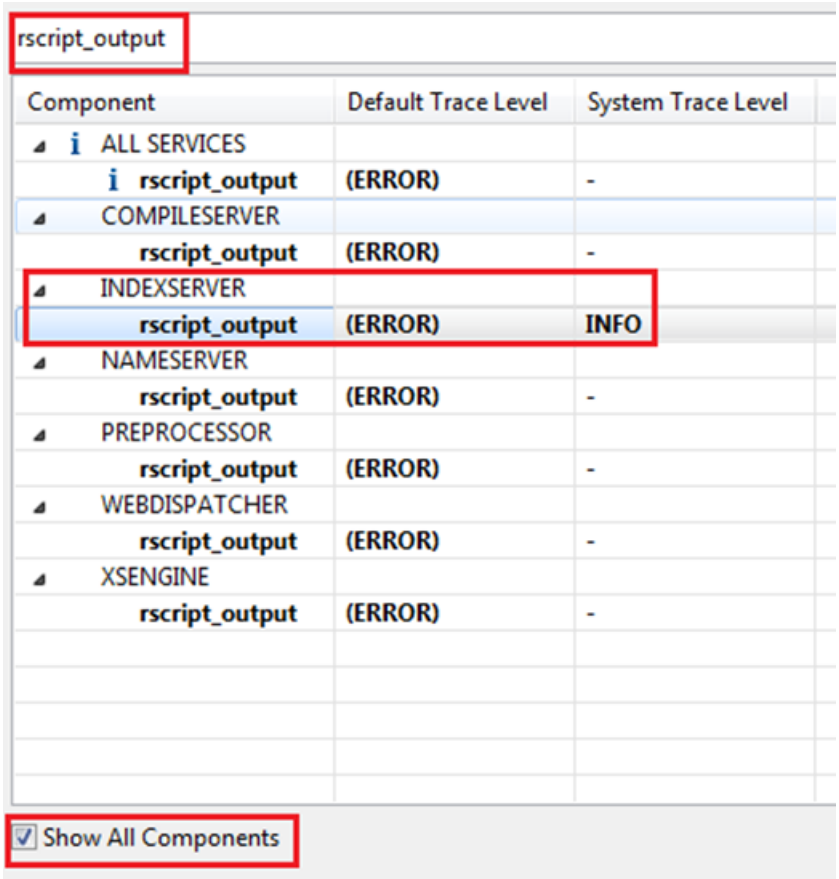
Perform the following steps to trace the R procedure output automatically.

Procedure

1. In the SAP HANA studio, open the *Administration Editor* and choose the *Trace Configuration* tab.
2. Locate `Database Trace` and click the *Edit Configuration* icon.



3. Select *Show All Components* and in the *Search* field, type **`rscript_output`**, then set the desired trace level.
   - If the trace level is set with `fatal`, `error`, or `warning`, the R script output cannot be traced automatically.
   - If the trace level is set with `info`, the R script output can be traced automatically.
   - If the trace level is set with `debug`, the R script output as well as part of the input and output variable data will be dumped automatically.
     You can set to this trace level to check roughly whether the data transfered from SAP HANA to Rserve has the right structure and content, and whether or not the result variable structure is correct.



Results

After the trace level of `rscript_output` is recorded, all the R script output are recorded and can be seen in the latest `indexserver_*.trc` file.

An example of R script follows:

Sample Code

```
DROP TYPE DUMMY_T;
CREATE TYPE DUMMY_T AS TABLE(
    ID INTEGER,
    VAL DOUBLE
);
DROP PROCEDURE USER_PROC;
CREATE PROCEDURE USER_PROC (IN input1 DUMMY_T, OUT result DUMMY_T)
LANGUAGE RLANG AS
BEGIN
    result <- rbind(input1, data.frame(ID=c(3,4), VAL=c(3.3,4.4)));    #add
two more rows to the result
    print("Hello World");    #this will generate stdout content, which will be
traced
    try(stop("ErrorMsg"));    #this will generate stderr content, which will
be traced
END;
--Prepare the input table
DROP TABLE DUMMY_INPUT_TBL;
CREATE COLUMN TABLE DUMMY_INPUT_TBL LIKE DUMMY_T;
INSERT INTO DUMMY_INPUT_TBL VALUES (1, 1.1);
INSERT INTO DUMMY_INPUT_TBL VALUES (2, 2.2);
CALL USER_PROC(DUMMY_INPUT_TBL, ?);
```

After the script is run, if the trace level of `rscript_output` is set to `info`, then open the latest `indexserver_*.trc` trace file, and the R script output appears as follows:

```
[31162]{215678}[28/7925] [redacted] i rscript_output   RserveConnection.cpp(00693) : R stdout content:
[1] "Hello World"
[31162]{215678}[28/7925] [redacted] e rscript_output   RserveConnection.cpp(00694) : R stderr content:
Error in try(stop("ErrorMsg")) : ErrorMsg
```

If the trace level of `rscript_output` is set to `debug`, then open the latest `indexserver_*.trc` trace file, and the R script output appears as follows:

```
[31351]{215678}[28/17265] [redacted] i rscript_output   RserveConnection.cpp(00693) : R stdout content:
[1] "input1"
'data.frame':   2 obs. of  2 variables:        dump input
$ ID : int  1 2
$ VAL: num  1.1 2.2
[1] "Hello World"
[1] "result"
'data.frame':   4 obs. of  2 variables:        dump output
$ ID : num  1 2 3 4
$ VAL: num  1.1 2.2 3.3 4.4
[31351]{215678}[28/17265] [redacted] e rscript_output   RserveConnection.cpp(00694) : R stderr content:
Error in try(stop("ErrorMsg")) : ErrorMsg
```

Note

If there are several lines of output, from both `stdout` and `stderr` interleafed, you see only two records in the trace file for one procedure call: one record contains all the `stdout` output and the other contains all the `stderr` output.

5.1.2 Trace R Procedure Output Manually

Set up an output trace to debug an R procedure manually.

Context

Sometimes, you might not have the privilege to set the right trace level of `rscript_output` component to see the traced output. Other times, you might prefer to have the R script output as an output variable in R procedure in order to extract only a part of the R script output. In these cases you need to write some extra code in your R scripts to manually trace the output of R procedure, saving it as an output variable in an R procedure.

Perform the following steps to trace the R procedure manually.

Procedure

1. Using an existing script, create two separate text connection objects, one for recording `stdout` and the other for `stderr`.

   > **Note**
   >
   > Additionally, you need to define a new output variable in the SQLScript procedure definition: `OUT rscripttrace OUTPUT_TRACE_T`

2. Use the `sink()` function in R language to redirect the output contents to the appropriate text connection objects.
3. After the content is captured, stop the redirection and assemble the tracing output variable.

   The following example shows you how to do this. Use the original script from which you want to get the `stdout/stderr` and add the lines between “# >>” and “# <<” to the R procedure.

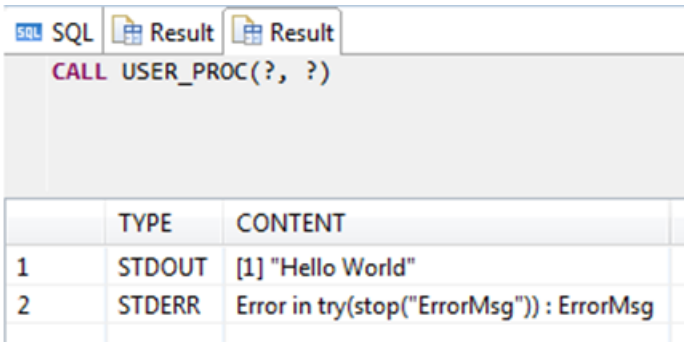
   Sample Code

```
DROP TYPE DUMMY_T;
CREATE TYPE DUMMY_T AS TABLE(
    ID INTEGER,
    VAL DOUBLE
);
DROP TYPE OUTPUT_TRACE_T;
CREATE TYPE OUTPUT_TRACE_T AS TABLE(
    TYPE VARCHAR(8),
    CONTENT VARCHAR(5000)
);
DROP PROCEDURE USER_PROC;
CREATE PROCEDURE USER_PROC (OUT result DUMMY_T, OUT rscripttrace
OUTPUT_TRACE_T)
LANGUAGE RLANG AS
BEGIN
    ############################
    # >>
```

```
    # create two separate text connection objects, one for recording
    # stdout and the other for stderr.  Then, they use the 'sink()'
    # function in R language to redirect the output contents to those
    # text connection objects.
    stdoutConn <- textConnection('stdoutContent', 'w', local=TRUE);
    stderrConn <- textConnection('stderrContent', 'w', local=TRUE);
    #redirect the stdout content
    sink(stdoutConn, type='output', split=TRUE);
    #redirect the stderr content
    sink(stderrConn, type='message');
    # <<
    ############################
    #Here is the script to manually record the output content
    # into output variables
    result <- data.frame(ID=c(1,2), VAL=c(1.1,2.2));
    print("Hello World");    #this will generate stdout content
    try(stop("ErrorMsg"));    #this will generate stderr content
    #Stop manually recording output
    ############################
    # >>
    # After the content is captured, users can stop the redirecting,
    # and assemble the tracing output variable.
    sink(NULL, type='output');    #stop redirecting stdout content
    sink(NULL, type='message');    #stop redirecting stderr content
    close(stdoutConn);    #cleanup
    close(stderrConn);    #cleanup
    rscripttrace <- rbind(
        data.frame(TYPE=character(0), CONTENT=character(0)),
        if (length(stdoutContent) > 0) data.frame(TYPE='STDOUT',
CONTENT=stdoutContent, stringsAsFactors=F) else NULL,
        if (length(stderrContent) > 0) data.frame(TYPE='STDERR',
CONTENT=stderrContent, stringsAsFactors=F) else NULL
    );    #assemble trace output variable
    # <<
    ############################
END;
CALL USER_PROC(?, ?);
```

Results

When the code is executed, the manually traced output appears in SAP HANA studio as a table:



> **Note**
>
> If the `rscript_output` trace level has been set to `info` or `debug`, which means the output is captured in the trace file already, and you manually redirect the output in the R script, then you need to bear the following points in mind:
>
> - When sinking `stdout` contents, SAP recommends that you add the `split=TRUE` option. Doing this means that the redirected output contents are recorded in the trace file as well. If you do not use this option, then you can only see the manually redirected contents in the output variable and not in trace files.
> - Be careful when sinking `stderr` contents. The redirected `stderr` contents are **not** recorded in the trace file, even if you stop manually redirecting the `stderr` contents.

5.2 Trace Detailed Execution Time of R Procedures

Context

To be able to see how much of the elapsed time during the R procedure execution was spent within the R environment and how much time was spent for the data transfer between SAP HANA and R you need to change the trace level for the SAP HANA-based R-Client.

Procedure

1. In the SAP HANA studio, go to *Administration Editor* > *Configuration tab* > *indexserver.ini* > *trace*.
2. Right click on *trace*, select *Add Parameter*, and add the following parameter:
   a. KEY: **`rclient`**
   b. VALUE: **`info`**

Results

Traces will then be available under *Administration Editor* > *Diagnosis Files tab* > *indexserver_*.trc*.

5.3 Measuring Performance in R

R provides some basic profiling functions to keep track of performance and memory consumption. If you plan to profile the R application (or use profiling functions like `tracemem()`), R has to be configured with the following command:

```
--enable-memory-profiling
```

- `system.time()` can be used to measure the total evaluation time for a given R expression. Set `gcFirst=TRUE` for garbage collection.
- `replicate()` can be used to average over invocations.
- `identical()` and `all.equal()` can be used to ensure that optimizations produce correct results.
- `Rprof()` can be used to profile multiple function calls and their execution time.
- `tracemem()` can be used to profile the memory consumption of the R environment.

5.4 Profiling R Procedures with Rprof

You can use the R function `Rprof()` to profile your R procedure.

```
DROP TABLE "Rprof";
CREATE COLUMN TABLE "Rprof"("function" VARCHAR(5000), "total.time" DOUBLE,
"total.pct" DOUBLE, "self.time" DOUBLE, "self.pct" DOUBLE);
DROP PROCEDURE USE_RPROF;
CREATE PROCEDURE USE_RPROF(OUT result "Rprof")
LANGUAGE RLANG AS
BEGIN
        tmp <- tempfile()
Rprof(tmp)
#--############################
# Your R code goes here…
#--############################
Rprof(NULL)
profile <- as.data.frame(summaryRprof(tmp)$by.total)
unlink(tmp)
result <- cbind("function"=rownames(profile), profile)
END;
CALL USE_RPROF("Rprof") WITH OVERVIEW;
SELECT * FROM "Rprof";
```

Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of willful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).

go.sap.com/registration/contact.html

© 2016 SAP SE or an SAP affiliate company. All rights reserved.
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.
Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.
These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.
SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.
Please see http://www.sap.com/corporate-en/legal/copyright/index.epx for additional trademark information and notices.