# Java API

Backend API for building new codeunits

# Sessions

For interacting with Tempus Serva records you need a session.
You can create one like this:

```
Session session = SessionFactory.getSession(this);
// or, in a codeunit
Session session = getSession();
```

All sessions need to be terminated after use (releasing DB connections etc.)

```
session.close();
// or in a finally (tests for NPE)
DbClose.close(session);
// or use try-with
try (Session session = SessionFactory.getSession(this)) {
} catch (TsCloseObjectRequired ignore) {}
```

All sessions are automatically closed after Policy#apiSessionMaxLifetime (default is 30 seconds).
If you suspect your code might be running for more than that, the session can be marked as *long running*.
This can be done durring the initialization of the session, or after the fact.

```
Session session = SessionFactory.getSession(context, this, 30);
// or, in a codeunit
Session session = getSession(30);
// or, after the fact
session.setLongRunningSession(30);
```

The above code extends the lifetime of the session by 30 seconds.

If you forget to close a session, an entry will be added to the eventlog, like this:

`[ClassName] probably forgot to close session`

# Security

# Context

# Constructing queries

First of create a query object

```
SolutionQuery myQuery = session.getSolutionQuery("mynewsolution");
// or (as of version 11230)
SolutionQuery myQuery = session.query("mynewsolution");
```

A query is built much like an SQL query by appending different fields to be SELECT'ed

```
myQuery.addSelectField("MYNUMBER");
myQuery.addSelectField("MYOTHERNUMBER");
// or (as of version 9336)
myQuery.addSelectField("MYNUMBER", "MYOTHERNUMBER");
// or (as of version 11230)
myQuery.select("MYNUMBER", "MYOTHERNUMBER");
```

Additionally WHERE components are added

```
myQuery.addWhereCriterion("StatusID", "In progress");
myQuery.addWhereCriterion("INCOME", QueryPart.MORE, 22);
myQuery.addWhereInList("PARENTITEMS", arrayListOfDataID);
// or (as of version 7347)
myQuery
  .addWhereCriterion("StatusID", "In progress")
  .addWhereCriterion("INCOME", QueryPart.MORE, 22)
  .addWhereInList("PARENTITEMS", arrayListOfDataID);
// or (as of version 11218)
myQuery
  .where("StatusID", "In progress")
  .where("INCOME", QueryPart.MORE, 22)
  .whereIn("PARENTITEMS", arrayListOfDataID);
```

Likewise operators can be added

```
myQuery.addWhereOR();


myQuery.addWhereAND();
```

```
// or (as of version 11218)
myQuery.or();


myQuery.and();
```

Note that lookup values etc. will be translated silently.

```
myQuery.setSortOrder("INCOME", true);
// or (as of version 11218)
myQuery.sortBy("INCOME", true);
```

Order records by income in descending order (true).

By default the API limits solution queries to 100 rows. To increase this:

```
myQuery.setLimit(100000);
// or (as of version 11218)
myQuery.limit(100000);
```

To do pagination use this:

```
myQuery.setLimitAndOffset(100, 100);
// or (as of version 11218)
myQuery.limitAndOffset(100, 100);
```

Finally execute the query and return the results

```
SolutionQueryResultSet records = myQuery.executeQuery();
// or (as of version 11218)
SolutionQueryResultSet records = myQuery.execute();
```

SolutionQuery supports method chaining, so the following two samples are equivalent.

```
SolutionQuery sq = session.getSolutionQuery("cars");
sq.addSelectField("MANUFACTURER", "MODEL");
sq.addWhereCriterion("PRICE", QueryPart.MORE, 250000);
sq.addWhereOR();
sq.addWhereCriterion("ONSALE", true);
sq.setSortOrderAsc("PRICE");
SolutionQueryResultSet rs = sq.executeQuery();
// or (as of version 7347)
```

```
SolutionQueryResultSet rs = session.getSolutionQuery("cars")
  .addSelectField("MANUFACTURER", "MODEL")
  .addWhereCriterion("PRICE", QueryPart.MORE, 250000)
  .addWhereOR()
  .addWhereCriterion("ONSALE", true)
  .setSortOrderAsc("PRICE")
  .executeQuery();
// or (as of version 11218)
SolutionQueryResultSet rs = session.query("cars")
  .select("MANUFACTURER", "MODEL")
  .where("PRICE", QueryPart.MORE, 250000)
  .or()
  .where("ONSALE", true)
  .sortByAsc("PRICE")
  .execute();
```

# Retrieve data

Data are extracted from the query resultset using a relative reference to the record number

```
for (int i = 0; i < rs.size(); i++) {
  String text = "Value nr " + i " = " + rs.getRecordValue(i, "MYVALUE");
  System.out.println(text);
}
// or
for (int i = 0; i < rs.size(); i++) {
  SolutionRecord sr = rs.getRecord(i);
  String text = "Value nr " + i " = " + sr.getValue("MYVALUE");
  System.out.println(text);
}
// or (as of version 9444)
for (SolutionRecord sr : rs.list()) {
  String text = "Value DataID " + sr.getDataID() " = " + sr.getValue("MYVALUE");
  System.out.println(text);
}
// or (as of version 11230)
SolutionRecord sr;
while ((sr = rs.next()) != null) {
  String text = "Value DataID " + sr.getDataID() " = " + sr.getValue("MYVALUE");
  System.out.println(text);
}
```

Retriever methods exist for different datatypes, try using the auto-suggestions in your IDE.

Up until version 11198 SolutionQueryResultSet wasn't memory optimized, storing all loaded SolutionRecods in memory.
Using rs.list() still loads all SolutionRecods into memory, and thus is the least optimized way to iterate over the ResultSet.

# Change / create data

# Code example

The following example shows how data is retrieved from af collection of templates and used to create a collection of records (with values copied from the template).

```
String thisKeyToParentTemplate = "VALGTSKABELON"; // Entity
String templateSolutionName = "tasktemplate"; // Entity
ArrayList<String> fieldsTemplate = new ArrayList(String [] {"TASK","DEADLINE","RESPONSIBLE"}); // list of fields
String templateKeyToParentTemplate = "SKABELON"; // Entity
String instanceSolutionName = "taskinstance";
String instanceKeyToParent = "LIST";
```

Now to the code ...

```
int parentTemplateDataID =
Parser.getInteger(c.fields.getElementByFieldName(thisKeyToParentTemplate).FieldValue);

try (Session session = SessionFactory.getSession(this)) {
  //Get data
  SolutionQuery opsaetning = session.getSolutionQuery(templateSolutionName);
  for (int i = 0; i < fieldsTemplate.size(); i++)
    opsaetning.addSelectField(fieldsTemplate.get(i));
  opsaetning.addWhereCriterion(templateKeyToParentTemplate, parentTemplateDataID);
  SolutionQueryResultSet recordsToCopy = opsaetning.executeQuery();

  int recordCount = recordsToCopy.size();
  for (int i = 0; i < recordCount; i++) {
    SolutionRecordNew instance = session.getSolutionRecordNew(instanceSolutionName);

    for (int fi = 0; fi < fieldsTemplate.size(); fi++) {
      String fieldNameToCopy = fieldsTemplate.get(fi);
      String value = recordsToCopy.getRecordValue(i, fieldNameToCopy);
      instance.setValue(fieldNameToCopy, value);
    }
    instance.setValueInteger(instanceKeyToParent, c.DataID);
    if (Parser.isNotEmpty(instanceKeyToTemplate))
```

```
      instance.setValueInteger(instanceKeyToTemplate, recordsToCopy.getReference(i));

    instance.persistChanges();
  }
} catch(Exception e) {
    e.printStackTrace();
} catch (TsCloseObjectRequired ex) {
}
```

The newer version (as of version 11230)

```
int parentTemplateDataID =
Parser.getInteger(c.fields.getElementByFieldName(thisKeyToParentTemplate).FieldValue);

try (Session session = SessionFactory.getSession(this)) {
  //Get data
  SolutionQueryResultSet recordsToCopy = session
    .query(templateSolutionName)
    .select(fieldsTemplate)
    .where(templateKeyToParentTemplate, parentTemplateDataID)
    .execute();

  SolutionRecord original;
  while ((original = recordsToCopy.next()) != null) {
    SolutionRecordNew instance = session.getSolutionRecordNew(instanceSolutionName);

    for (String field : fieldsTemplate) {
      String value = original.getValue(field);
      instance.setValue(field, value);
    }
    instance.setValue(instanceKeyToParent, c.DataID);
    if (Parser.isNotEmpty(instanceKeyToTemplate))
      instance.setValue(instanceKeyToTemplate, original.getDataID());

    instance.persistChanges();
  }
} catch(Exception ex) {
    ex.printStackTrace();
} catch (TsCloseObjectRequired ignore) {
}
```

# Command

# DbConnection

# EventHandler