
PyAPplus64

Release 1.0.1

Thomas Tuerk

06.05.2023

Inhaltsverzeichnis

1	Beschreibung	1
2	typische Anwendungsfälle	3
3	Abhängigkeiten	7
4	Beispiele	9
5	PyAPplus64 package	21
	Python-Modulindex	45

Das Paket *PyAPplus64* enthält eine Sammlung von Python Tools für die Interaktion mit dem ERP-System APplus 6.4. Es sollte auch für andere APplus Versionen nützlich sein.

Zielgruppe sind APplus-Administratoren und Anpassungs-Entwickler. *PyAPplus64* erlaubt u.a.

- **einfacher Zugriff auf SOAP-Schnittstelle des APP-Servers**
 - damit Zugriff auf SysConfig
 - Zugriff auf Tools wie z.B. *nextNumber* für Erzeugung der nächsten Nummer für ein Business-Object
 - ...
- **Zugriff auf APplus DB per direktem DB-Zugriff und mittels SOAP**
 - automatischer Aufruf von *completeSQL*, um per AppServer SQL-Statements um z.B. Mandanten erweitern zu lassen
 - Tools für einfache Benutzung von *useXML*, d.h. für das Einfügen, Löschen und Ändern von Datensätzen mit Hilfe des APP-Servers. Genau wie bei Änderungen an Datensätzen über die Web-Oberfläche und im Gegensatz zum direkten Zugriff über die Datenbank werden dabei evtl. zusätzliche Checks ausgeführt, bestimmte Felder automatisch gesetzt oder bestimmte Aktionen angestoßen.
- **das Duplizieren von Datensätzen**
 - zu kopierende Felder aus XML-Definitionen werden ausgewertet
 - Abhängige Objekte können einfach ebenfalls mit-kopiert werden
 - Änderungen wie beispielsweise Nummer des Objektes möglich
 - Unterstützung für Kopieren dyn. Attribute
 - Anlage neuer Objekte mittels APP-Server
 - Datensätze können zwischen Systemen kopiert und auch in Dateien gespeichert werden
 - Beispiel: Kopieren von Artikeln mit Arbeitsplan und Stückliste zwischen Deploy- und Prod-System
- **einfaches Erstellen von Excel-Reports aus SQL-Abfragen**
 - mittels Pandas und XlsxWriter

- einfache Wrapper um andere Libraries, spart aber Zeit

- ...

In *PyAPplus64* wurden die Features (vielleicht leicht verallgemeinert) implementiert, die ich für konkrete Aufgabenstellungen benötigte. Ich gehe davon aus, dass im Laufe der Zeit weitere Features hinzukommen.

1.1 Warnung

PyAPplus64 erlaubt den schreibenden Zugriff auf die APplus Datenbank und beliebige Aufrufe von SOAP-Methoden. Unsachgemäße Nutzung kann Ihre Daten zerstören. Benutzen Sie *PyAPplus64* daher bitte vorsichtig. Zudem kann ich leider nicht garantieren, dass *PyAPplus64* fehlerfrei ist.

typische Anwendungsfälle

2.1 einfache Admin-Aufgaben

Selten auftretende Admin-Aufgaben lassen sich gut mittels Python-Skripten automatisieren. Es ist sehr einfach möglich, auf die DB, aber auch auf SOAP-Schnittstelle zuzugreifen. Ich habe dies vor allem für Wartungsarbeiten an Anpassungstabellen, die für eigene Erweiterungen entwickelt wurden, genutzt.

Als triviales Beispiel sucht folgender Code alle *DOCUMENTS* Einträge in Artikeln (angezeigt als *Bild* in *Artikel-Rec.aspx*), für die Datei, auf die verwiesen wird, nicht im Dateisystem existiert. Diese fehlenden Dateien werden ausgegeben und das Feld *DOCUMENTS* gelöscht. Das Löschen erfolgt dabei über *useXML*, so dass die Felder *UPDDATE* und *UPDUSER* korrekt gesetzt werden.

```
1 import pathlib
2 import PyAPplus64
3 import applus_configs
4 from typing import Optional
5
6
7 def main(confFile: pathlib.Path, updateDB: bool, docDir: Optional[str] = None) ->
  ↳None:
8     server = PyAPplus64.applus.applusFromConfigFile(confFile)
9
10    if docDir is None:
11        docDir = str(server.scripttool.getInstallPathWebServer().joinpath("DocLib"))
12
13    sql = PyAPplus64.sql_utils.SqlStatementSelect("ARTIKEL")
14    sql.addFields("ID", "ARTIKEL", "DOCUMENTS")
15    sql.where.addConditionFieldStringNotEmpty("DOCUMENTS")
16
17    for row in server.dbQueryAll(sql):
18        doc = pathlib.Path(docDir + row.DOCUMENTS)
19        if not doc.exists():
20            print("Bild '{}' für Artikel '{}' nicht gefunden".format(doc, row.
  ↳ARTIKEL))
21
22        if updateDB:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

23         upd = server.mkUseXMLRowUpdate("ARTIKEL", row.ID)
24         upd.addField("DOCUMENTS", None)
25         upd.update()
26
27
28 if __name__ == "__main__":
29     main(applus_configs.serverConfYamlTest, False)

```

Man kann alle Python Bibliotheken nutzen. Als Erweiterung wäre es in obigem Script zum Beispiel einfach möglich, alle BMP-Bilder zu suchen, nach PNG zu konvertieren und den DB-Eintrag anzupassen.

2.2 Ad-hoc Reports

APplus erlaubt es mittels Jasper-Reports, flexible und schöne Reports zu erzeugen. Dies funktioniert gut und ist für regelmäßig benutzte Reports sehr sinnvoll.

Für ad-hoc Reports, die nur sehr selten oder sogar nur einmal benutzt werden, ist die Erstellung eines Jasper-Reports und die Einbindung in APplus jedoch zu zeitaufwändig. Teilweise genügen die Ergebnisse einer SQL-Abfrage, die direkt im MS SQL Server abgesetzt werden kann. Wird es etwas komplizierter oder sollen die Ergebnisse noch etwas verarbeitet werden, bietet sich evtl. Python an.

Folgendes Script erzeugt zum Beispiel eine Excel-Tabelle, mit einer Übersicht, welche Materialien wie oft für Artikel benutzt werden:

```

1  import PyAPplus64
2  import applus_configs
3  import pathlib
4
5
6  def main(confFile: pathlib.Path, outfile: str) -> None:
7      server = PyAPplus64.applus.applusFromConfigFile(confFile)
8
9      # Einfache SQL-Anfrage
10     sql1 = ("select Material, count(*) as Anzahl from ARTIKEL "
11            "group by MATERIAL having MATERIAL is not null "
12            "order by Anzahl desc")
13     df1 = PyAPplus64.pandas.pandasReadSql(server, sql1)
14
15     # Sql Select-Statements können auch über SqlStatementSelect zusammgebaut
16     # werden. Die ist bei vielen, komplizierten Bedingungen teilweise hilfreich.
17     sql2 = PyAPplus64.SqlStatementSelect("ARTIKEL")
18     sql2.addFields("Material", "count(*) as Anzahl")
19     sql2.addGroupBy("MATERIAL")
20     sql2.having.addConditionFieldIsNotNull("MATERIAL")
21     sql2.order = "Anzahl desc"
22     df2 = PyAPplus64.pandas.pandasReadSql(server, sql2)
23
24     # Ausgabe als Excel mit 2 Blättern
25     PyAPplus64.pandas.exportToExcel(outfile, [(df1, "Materialien"), (df2,
26     ↪ "Materialien 2")], addTable=True)
27
28 if __name__ == "__main__":
29     main(applus_configs.serverConfYamlTest, "myout.xlsx")

```

Dieses kurze Script nutzt Standard-Pandas Methoden zur Erzeugung der Excel-Datei. Allerdings sind diese Metho-

den in den Aufrufen von *pandasReadSql* und *exportToExcel* gekapselt, so dass der Aufruf sehr einfach ist. Zudem ist es sehr einfach, die Verbindung zur Datenbank und zum APP-Server mittels der YAML-Konfigdatei herzustellen. Bei diesem Aufruf kann optional ein Nutzer und eine Umgebung übergeben werden, die die Standard-Werte aus der YAML-Datei überschreiben. *pandasReadSql* nutzt intern *completeSQL*, so dass der für die Umgebung korrekte Mandant automatisch verwendet wird.

2.3 Anbindung eigener Tools

Ursprünglich wurde *PyAPplus64* für die Anbindung einer APplus-Anpassung geschrieben. Diese Anpassung ist als Windows-Service auf einem eigenen Rechner installiert und überwacht dort ein bestimmtes Verzeichnis. Bei Änderungen an Dateien in diesem Verzeichnis (Hinzufügen, Ändern, Löschen) werden die Dateien verarbeitet und die Ergebnisse an APplus gemeldet. Dafür werden DB-Operationen aber auch SOAP-Calls benutzt. Ebenso wird auf die SysConf zugegriffen.

Viele solcher Anpassungen lassen sich gut und sinnvoll im App-Server einrichten und als Job regelmäßig aufrufen. Im konkreten Fall wird jedoch für die Verarbeitung der Dateien viel Rechenzeit benötigt. Dies würde dadurch verschlimmert, dass die Dateien nicht auf der gleichen Maschine wie der App-Server liegen und somit viele, relativ langsame Dateizugriffe über das Netzwerk nötig wären. Hinzu kommt, dass die Verarbeitung der Dateien dank existierender Bibliotheken in Python einfacher ist.

PyAPplus64 kann für solche Anpassungen eine interessante Alternative zur Implementierung im App-Server oder zur Entwicklung eines Tools ohne spezielle Bibliotheken sein. Nach Initialisierung des Servers:

```
import PyAPplus64

server = PyAPplus64.applus.applusFromConfigFile("my-applus-server.yaml")
```

bietet *P2APplus64* wie oben demonstriert einfachen lesenden und schreibenden Zugriff auf die DB. Hierbei werden automatisch die für die Umgebung nötigen Mandanten zu den SQL Statements hinzugefügt. Zudem ist einfach ein Zugriff auf die Sysconf möglich:

```
print (server.sysconf.getString("STAMM", "MYLAND"))
print (server.sysconf.getList("STAMM", "EULAENDER"))
```

Dank der Bibliothek *zeep* ist es auch sehr einfach möglich, auf beliebige SOAP-Methoden zuzugreifen. Beispielsweise kann auf die Sys-Config auch händisch, d.h. durch direkten Aufruf einer SOAP-Methode, zugegriffen werden:

```
client = server.server_conn.getClient("p2system", "SysConf");
print (client.service.getString("STAMM", "MYLAND"))
```


3.1 pyodbc

Für die Datenbankverbindung wird `pyodbc` (`python -m pip install pyodbc`) verwendet. Der passende ODBC Treiber, MS SQL Server 2012 Native Client, wird zusätzlich benötigt. Dieser kann von Microsoft bezogen werden.

3.2 zeep

Die Soap-Library `zeep` wird benutzt (`python -m pip install zeep`).

3.3 PyYaml

Die Library `pyyaml` wird für Config-Dateien benutzt (`python -m pip install pyyaml`).

3.4 Sphinx

Diese Dokumentation ist mit Sphinx geschrieben. `python -m pip install sphinx`. Dokumentation ist im Unterverzeichnis `docs` zu finden. Sie kann mittels `make.bat html` erzeugt werden, dies ruft intern `sphinx-build -M html source build` auf. Die Dokumentation der Python-API sollte evtl. vorher mittels `sphinx-apidoc -T -f ../src/PyAPplus64 -o source/generated` erzeugt oder aktualisiert werden. Evtl. können 2 Aufrufe von `make.bat html` sinnvoll sein, falls sich die Struktur der Dokumentation ändert. Diese Aufrufe werden von `builddocs.sh` automatisiert.

Die erzeugte Doku findet sich im Verzeichnis `build/html`.

3.5 Pandas / SQLAlchemy / xlsxwriter

Sollen Excel-Dateien mit Pandas erzeugt, werden, so muss Pandas, SQLAlchemy und xlsxwriter installiert sein (*python -m pip install pandas sqlalchemy xlsxwriter*).

Im Verzeichnis `examples` finden sich Python Dateien, die die Verwendung von *PyAPplus64* demonstrieren.

4.1 Config-Dateien

Viele Scripte teilen sich Einstellungen. Beispielsweise greifen fast alle Scripte irgendwie auf APplus zu und benötigen Informationen, mit welchem APP-Server, welchem Web-Server und welcher Datenbank sie sich verbinden sollen. Solche Informationen, insbesondere die Passwörter, werden nicht in jedem Script gespeichert, sondern nur in den Config-Dateien. Es bietet sich wohl meist an, 3 Konfigdateien zu erstellen, je eine für das Deploy-, das Test- und das Prod-System. Ein Beispiel ist im Unterverzeichnis `examples/applus-server.yaml` zu finden.

```
1 # Einstellung für die Verbindung mit dem APP-, Web- und DB-Server.
2 # Viele der Einstellungen sind im APplus Manager zu finden
3
4 appserver : {
5     server : "some-server",
6     port : 2037,
7     user : "asol.projects",
8     env : "default-umgebung" # hier wirklich Umgebung, nicht Mandant verwenden
9 }
10 webserver : {
11     baseurl : "http://some-server/APplusProd6/"
12 }
13 dbserver : {
14     server : "some-server",
15     db : "APplusProd6",
16     user : "SA",
17     password : "your-db-password"
18 }
```

Damit nicht in jedem Script immer wieder neu die Konfig-Dateien ausgewählt werden müssen, werden die Konfigs für das Prod-, Test- und Deploy-System in `examples/applus_configs.py` hinterlegt. Diese Datei wird in allen Scripten importiert, so dass das Config-Verzeichnis und die darin enthaltenen Configs einfach zur Verfügung stehen.

```

1 import pathlib
2
3 basedir = pathlib.Path(__file__)
4 configdir = basedir.joinpath("config")
5
6 serverConfYamlDeploy = configdir.joinpath("applus-server-deploy.yaml")
7 serverConfYamlTest = configdir.joinpath("applus-server-test.yaml")
8 serverConfYamlProd = configdir.joinpath("applus-server-prod.yaml")

```

4.2 read_settings.py

Einfaches Beispiel für Auslesen der SysConf und bestimmter Einstellungen.

```

1 # Einfaches Script, das verschiedene Werte des Servers ausliest.
2 # Dies sind SysConfig-Einstellungen, aber auch der aktuelle Mandant,
3 # Systemnamen, ...
4
5 import pathlib
6 import PyAPplus64
7 import applus_configs
8 from typing import Optional, Union
9
10
11 def main(confFile: Union[str, pathlib.Path], user: Optional[str] = None, env:
12 ↪ Optional[str] = None) -> None:
13     server = PyAPplus64.applusFromConfigFile(confFile, user=user, env=env)
14
15     print("\n\nSysConf Lookups:")
16
17     print("  Default Auftragsart:", server.sysconf.getString("STAMM",
18 ↪ "DEFAULTAUFTRAGSART"))
19     print("  Auftragsarten:")
20     arten = server.sysconf.getList("STAMM", "AUFTRAGSART", sep='\n')
21     if not arten:
22         arten = []
23     for a in arten:
24         print("    - " + a)
25
26     print("  Firmen-Nr. automatisch vergeben:", server.sysconf.getBoolean("STAMM",
27 ↪ "FIRMAAUTOMATIK"))
28     print("  Anzahl Artikelstellen:", server.sysconf.getInt("STAMM",
29 ↪ "ARTKLASSIFNRLAENGE"))
30
31     print("\n\nScriptTool:")
32
33     print("  CurrentDate:", server.scripttool.getCurrentDate())
34     print("  CurrentTime:", server.scripttool.getCurrentTime())
35     print("  CurrentDateTime:", server.scripttool.getCurrentDateTime())
36     print("  LoginName:", server.scripttool.getLoginName())
37     print("  UserName:", server.scripttool.getUserName())
38     print("  UserFullName:", server.scripttool.getUserFullName())
39     print("  SystemName:", server.scripttool.getSystemName())
40     print("  Mandant:", server.scripttool.getMandant())
41     print("  MandantName:", server.scripttool.getMandantName())
42     print("  InstallPath:", server.scripttool.getInstallPath())

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

39     print("  InstallPathAppServer:", server.scripttool.getInstallPathAppServer())
40     print("  InstallPathWebServer:", server.scripttool.getInstallPathWebServer())
41     print("  ServerInfo - Version:", server.scripttool.getServerInfo().find("version
↪").text)
42
43
44 if __name__ == "__main__":
45     main(applus_configs.serverConfYamlTest)

```

4.3 check_dokumente.py

Einfaches Beispiel für lesenden und schreibenden Zugriff auf APplus Datenbank.

```

1  import pathlib
2  import PyAPplus64
3  import applus_configs
4  from typing import Optional
5
6
7  def main(confFile: pathlib.Path, updateDB: bool, docDir: Optional[str] = None) ->
↪None:
8      server = PyAPplus64.applus.applusFromConfigFile(confFile)
9
10     if docDir is None:
11         docDir = str(server.scripttool.getInstallPathWebServer().joinpath("DocLib"))
12
13     sql = PyAPplus64.sql_utils.SqlStatementSelect("ARTIKEL")
14     sql.addFields("ID", "ARTIKEL", "DOCUMENTS")
15     sql.where.addConditionFieldStringNotEmpty("DOCUMENTS")
16
17     for row in server.dbQueryAll(sql):
18         doc = pathlib.Path(docDir + row.DOCUMENTS)
19         if not doc.exists():
20             print("Bild '{}' für Artikel '{}' nicht gefunden".format(doc, row.
↪ARTIKEL))
21
22         if updateDB:
23             upd = server.mkUseXMLRowUpdate("ARTIKEL", row.ID)
24             upd.addField("DOCUMENTS", None)
25             upd.update()
26
27
28 if __name__ == "__main__":
29     main(applus_configs.serverConfYamlTest, False)

```

4.4 adhoc_report.py

Sehr einfaches Beispiel zur Erstellung einer Excel-Tabelle aus einer SQL-Abfrage.

```

1 import PyAPplus64
2 import applus_configs
3 import pathlib
4
5
6 def main(confFile: pathlib.Path, outfile: str) -> None:
7     server = PyAPplus64.applus.applusFromConfigFile(confFile)
8
9     # Einfache SQL-Anfrage
10    sql1 = ("select Material, count(*) as Anzahl from ARTIKEL "
11           "group by MATERIAL having MATERIAL is not null "
12           "order by Anzahl desc")
13    df1 = PyAPplus64.pandas.pandasReadSql(server, sql1)
14
15    # Sql Select-Statements können auch über SqlStatementSelect zusammgebaut
16    # werden. Die ist bei vielen, komplizierten Bedingungen teilweise hilfreich.
17    sql2 = PyAPplus64.SqlStatementSelect("ARTIKEL")
18    sql2.addFields("Material", "count(*) as Anzahl")
19    sql2.addGroupBy("MATERIAL")
20    sql2.having.addConditionFieldIsNotNull("MATERIAL")
21    sql2.order = "Anzahl desc"
22    df2 = PyAPplus64.pandas.pandasReadSql(server, sql2)
23
24    # Ausgabe als Excel mit 2 Blättern
25    PyAPplus64.pandas.exportToExcel(outfile, [(df1, "Materialien"), (df2,
26    ↪ "Materialien 2")], addTable=True)
27
28 if __name__ == "__main__":
29    main(applus_configs.serverConfYamlTest, "myout.xlsx")

```

4.5 mengenabweichung.py

Etwas komplizierteres Beispiel zur Erstellung einer Excel-Datei aus SQL-Abfragen.

```

1 # Erzeugt Excel-Tabellen mit Werkstattaufträgen und Werkstattauftragspositionen mit
2 ↪ Mengenabweichungen
3
4 import datetime
5 import PyAPplus64
6 import applus_configs
7 import pandas as pd # type: ignore
8 import pathlib
9 from typing import Tuple, Union, Optional
10
11 def ladeAlleWerkstattauftragMengenabweichungen(
12     server: PyAPplus64.APplusServer,
13     cond: Union[PyAPplus64.SqlCondition, str, None] = None) -> pd.DataFrame:
14     sql = PyAPplus64.sql_utils.SqlStatementSelect("WAUFTRAG w")
15     sql.addLeftJoin("personal p", "w.UPDUSER = p.PERSONAL")

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

16 sql.addFieldTable("w", "ID", "BAUFTRAG", "POSITION")
17 sql.addFieldTable("w", "MENGE", "MENGE_IST",
18                 "APLAN as ARTIKEL", "NAME as ARTIKELNAME")
19 sql.addFieldTable("w", "MENGE", "MENGE_IST",
20                 "APLAN as ARTIKEL", "NAME as ARTIKELNAME")
21 sql.addFieldTable("w", "UPDDATE", "p.NAME as UPDNAME")
22
23 sql.where.addFieldGe("w.STATUS", 5)
24 sql.where.addCondition("abs(w.MENGE-w.MENGE_IST) > 0.001")
25 sql.where.addCondition(cond)
26 sql.order = "w.UPDDATE"
27 dfOrg = PyAPplus64.pandas.pandasReadSql(server, sql)
28
29 # Add Links
30 df = dfOrg.copy()
31 df = df.drop(columns=["ID"])
32 # df = df[['POSITION', 'BAUFTRAG', 'MENGE']] # reorder / filter columns
33
34 df['POSITION'] = PyAPplus64.pandas.mkHyperlinkDataframeColumn(
35     dfOrg,
36     lambda r: r.POSITION,
37     lambda r: server.makeWebLinkWauftrag(
38         baufrtag=r.BAUFTRAG, accessid=r.ID))
39 df['BAUFTRAG'] = PyAPplus64.pandas.mkHyperlinkDataframeColumn(
40     dfOrg,
41     lambda r: r.BAUFTRAG,
42     lambda r: server.makeWebLinkBauftrag(bauftrag=r.BAUFTRAG))
43
44 colNames = {
45     "BAUFTRAG": "Betriebsauftrag",
46     "POSITION": "Pos",
47     "MENGENABWEICHUNG": "Mengenabweichung",
48     "MENGE": "Menge",
49     "MENGE_IST": "Menge-Ist",
50     "ARTIKEL": "Artikel",
51     "ARTIKELNAME": "Artikel-Name",
52     "UPDDATE": "geändert am",
53     "UPDNAME": "geändert von"
54 }
55 df.rename(columns=colNames, inplace=True)
56
57 return df
58
59
60 def ladeAlleWerkstattauftragPosMengenabweichungen(
61     server: PyAPplus64.APplusServer,
62     cond: Union[PyAPplus64.SqlCondition, str, None] = None) -> pd.DataFrame:
63     sql = PyAPplus64.sql_utils.SqlStatementSelect("WAUFTRAGPOS w")
64     sql.addLeftJoin("personal p", "w.UPDUSER = p.PERSONAL")
65
66     sql.addFieldTable("w", "ID", "BAUFTRAG", "POSITION", "AG")
67     sql.addFieldTable("w", "MENGE", "MENGE_IST", "APLAN as ARTIKEL")
68     sql.addFieldTable("w", "MENGE", "MENGE_IST", "APLAN as ARTIKEL")
69     sql.addFieldTable("w", "UPDDATE", "p.NAME as UPDNAME")
70
71     sql.where.addFieldEq("w.STATUS", 4)

```

(Fortsetzung auf der nächsten Seite)

```

72 sql.where.addCondition("abs(w.MENGE-w.MENGE_IST) > 0.001")
73 sql.where.addCondition(cond)
74 sql.order = "w.UPDDATE"
75
76 dfOrg = PyAPplus64.pandas.pandasReadSql(server, sql)
77
78 # Add Links
79 df = dfOrg.copy()
80 df = df.drop(columns=["ID"])
81 df['POSITION'] = PyAPplus64.pandas.mkHyperlinkDataframeColumn(
82     dfOrg,
83     lambda r: r.POSITION,
84     lambda r: server.makeWebLinkWauftrag(
85         baufrtrag=r.BAUFTRAG, accessid=r.ID))
86 df['BAUFTRAG'] = PyAPplus64.pandas.mkHyperlinkDataframeColumn(
87     dfOrg,
88     lambda r: r.BAUFTRAG,
89     lambda r: server.makeWebLinkBauftrag(bauftrag=r.BAUFTRAG))
90 df['AG'] = PyAPplus64.pandas.mkHyperlinkDataframeColumn(
91     dfOrg,
92     lambda r: r.AG,
93     lambda r: server.makeWebLinkWauftragPos(
94         baufrtrag=r.BAUFTRAG, position=r.POSITION, accessid=r.ID))
95
96 # Demo zum Hinzufügen einer berechneten Spalte
97 # df['BAUFPOSAG'] = PyAPplus64.pandas.mkDataframeColumn(dfOrg,
98 #     lambda r: "{}.{} AG {}".format(r.BAUFTRAG, r.POSITION, r.
99 ↪AG))
100
101 # Rename Columns
102 colNames = {
103     "BAUFTRAG": "Betriebsauftrag",
104     "POSITION": "Pos",
105     "AG": "AG",
106     "MENGENABWEICHUNG": "Mengenabweichung",
107     "MENGE": "Menge",
108     "MENGE_IST": "Menge-Ist",
109     "ARTIKEL": "Artikel",
110     "UPDDATE": "geändert am",
111     "UPDNAME": "geändert von"
112 }
113 df.rename(columns=colNames, inplace=True)
114 return df
115
116 def computeInYearMonthCond(field: str, year: Optional[int] = None,
117     month: Optional[int] = None) -> Optional[PyAPplus64.
118 ↪SqlCondition]:
119     if not (year is None):
120         if month is None:
121             return PyAPplus64.sql_utils.SqlConditionDateTimeFieldInYear(field, year)
122         else:
123             return PyAPplus64.sql_utils.SqlConditionDateTimeFieldInMonth(field, year, ↪
124 ↪month)
125     else:
126         return None

```

(Fortsetzung der vorherigen Seite)

```

125
126
127 def computeFileName(year: Optional[int] = None, month: Optional[int] = None) -> str:
128     if year is None:
129         return 'mengenabweichungen-all.xlsx'
130     else:
131         if month is None:
132             return 'mengenabweichungen-{:04d}.xlsx'.format(year)
133         else:
134             return 'mengenabweichungen-{:04d}-{:02d}.xlsx'.format(year, month)
135
136
137 def _exportInternal(server: PyAPplus64.APplusServer, fn: str,
138                   cond: Union[PyAPplus64.SqlCondition, str, None]) -> int:
139     df1 = ladeAlleWerkstattauftragMengenabweichungen(server, cond)
140     df2 = ladeAlleWerkstattauftragPosMengenabweichungen(server, cond)
141     print("erzeuge " + fn)
142     PyAPplus64.pandas.exportToExcel(fn, [(df1, "WAuftrag"), (df2, "WAuftrag-Pos")],
143     ↪addTable=True)
144     return len(df1.index) + len(df2.index)
145
146 def exportVonBis(server: PyAPplus64.APplusServer, fn: str,
147                 von: Optional[datetime.datetime], bis: Optional[datetime.datetime]) -
148     ↪ int:
149     cond = PyAPplus64.sql_utils.SqlConditionDateTimeFieldInRange("w.UPDDATE", von,
150     ↪bis)
151     return _exportInternal(server, fn, cond)
152
153 def exportYearMonth(server: PyAPplus64.APplusServer,
154                   year: Optional[int] = None, month: Optional[int] = None) -> int:
155     cond = computeInYearMonthCond("w.UPDDATE", year=year, month=month)
156     fn = computeFileName(year=year, month=month)
157     return _exportInternal(server, fn, cond)
158
159 def computePreviousMonthYear(cyear: int, cmonth: int) -> Tuple[int, int]:
160     if cmonth == 1:
161         return (cyear-1, 12)
162     else:
163         return (cyear, cmonth-1)
164
165
166 def computeNextMonthYear(cyear: int, cmonth: int) -> Tuple[int, int]:
167     if cmonth == 12:
168         return (cyear+1, 1)
169     else:
170         return (cyear, cmonth+1)
171
172
173 def main(confFile: Union[str, pathlib.Path], user: Optional[str] = None, env:
174     ↪Optional[str] = None) -> None:
175     server = PyAPplus64.applusFromConfigFile(confFile, user=user, env=env)
176
177     now = datetime.date.today()

```

(Fortsetzung auf der nächsten Seite)

```

177 (cmonth, cyear) = (now.month, now.year)
178 (pyear, pmonth) = computePreviousMonthYear(cyear, cmonth)
179
180 # Ausgaben
181 exportYearMonth(server, cyear, cmonth) # Aktueller Monat
182 exportYearMonth(server, pyear, pmonth) # Vorheriger Monat
183 # export(cyear) # aktuelles Jahr
184 # export(cyear-1) # letztes Jahr
185 # export() # alles
186
187
188 if __name__ == "__main__":
189     main(applus_configs.serverConfYamlTest)

```

4.6 mengenabweichung_gui.py

Beispiel für eine sehr einfache GUI, die die Eingabe einfacher Parameter erlaubt. Die GUI wird um die Erzeugung von Excel-Dateien mit Mengenabweichungen gebaut.

```

1 import PySimpleGUI as sg # type: ignore
2 import mengenabweichung
3 import datetime
4 import PyAPplus64
5 import applus_configs
6 import pathlib
7 from typing import Tuple, Optional, Union
8
9
10 def parseDate(dateS: str) -> Tuple[Optional[datetime.datetime], bool]:
11     if dateS is None or dateS == '':
12         return (None, True)
13     else:
14         try:
15             return (datetime.datetime.strptime(dateS, '%d.%m.%Y'), True)
16         except:
17             sg.popup_error("Fehler beim Parsen des Datums '{}'.format(dateS))
18             return (None, False)
19
20
21 def createFile(server: PyAPplus64.APplusServer, fileS: str, vonS: str, bisS: str) ->
↳ None:
22     (von, vonOK) = parseDate(vonS)
23     if not vonOK:
24         return
25
26     (bis, bisOK) = parseDate(bisS)
27     if not bisOK:
28         return
29
30     if (fileS is None) or fileS == '':
31         sg.popup_error("Es wurde keine Ausgabedatei ausgewählt.")
32         return
33     else:
34         file = pathlib.Path(fileS)

```

(Fortsetzung der vorherigen Seite)

```

35
36     c = mengenabweichung.exportVonBis(server, file.as_posix(), von, bis)
37     sg.popup_ok("{} Datensätze erfolgreich in Datei '{}' geschrieben.".format(c, ↵
↵file))
38
39
40 def main(confFile: Union[str, pathlib.Path], user: Optional[str] = None, env: ↵
↵Optional[str] = None) -> None:
41     server = PyAPplus64.applusFromConfigFile(confFile, user=user, env=env)
42
43     layout = [
44         [sg.Text(('Bitte geben Sie an, für welchen Zeitraum die '
45                 'Mengenabweichungen ausgegeben werden sollen:'))],
46         [sg.Text('Von (einschließlich)', size=(15, 1)), sg.InputText(key='Von'),
47          sg.CalendarButton("Kalender", close_when_date_chosen=True,
48                           target="Von", format='%d.%m.%Y')],
49         [sg.Text('Bis (ausschließlich)', size=(15, 1)), sg.InputText(key='Bis'),
50          sg.CalendarButton("Kalender", close_when_date_chosen=True,
51                           target="Bis", format='%d.%m.%Y')],
52         [sg.Text('Ausgabedatei', size=(15, 1)), sg.InputText(key='File'),
53          sg.FileSaveAs(button_text="wählen",
54                       target="File",
55                       file_types=(('Excel Files', '*.xlsx'),),
56                       default_extension=".xlsx")],
57         [sg.Button("Aktueller Monat"), sg.Button("Letzter Monat"),
58          sg.Button("Aktuelles Jahr"), sg.Button("Letztes Jahr")],
59         [sg.Button("Speichern"), sg.Button("Beenden")]
60     ]
61
62     systemName = server.scripttool.getSystemName() + "/" + server.scripttool.
↵getMandant()
63     window = sg.Window("Mengenabweichung " + systemName, layout)
64     now = datetime.date.today()
65     (cmonth, cyear) = (now.month, now.year)
66     (pyear, pmonth) = mengenabweichung.computePreviousMonthYear(cyear, cmonth)
67     (nyear, nmonth) = mengenabweichung.computeNextMonthYear(cyear, cmonth)
68
69     while True:
70         event, values = window.read()
71         if event == sg.WIN_CLOSED or event == 'Beenden':
72             break
73         if event == 'Aktueller Monat':
74             window['Von'].update(value="01.{:02d}.{:04d}".format(cmonth, cyear))
75             window['Bis'].update(value="01.{:02d}.{:04d}".format(nmonth, nyear))
76         if event == 'Letzter Monat':
77             window['Von'].update(value="01.{:02d}.{:04d}".format(pmonth, pyear))
78             window['Bis'].update(value="01.{:02d}.{:04d}".format(cmonth, cyear))
79         if event == 'Aktuelles Jahr':
80             window['Von'].update(value="01.01.{:04d}".format(cyear))
81             window['Bis'].update(value="01.01.{:04d}".format(cyear+1))
82         if event == 'Letztes Jahr':
83             window['Von'].update(value="01.01.{:04d}".format(cyear-1))
84             window['Bis'].update(value="01.01.{:04d}".format(cyear))
85         if event == 'Speichern':
86             try:
87                 createFile(server, values.get('File', None),

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```

88         values.get('Von', None), values.get('Bis', None))
89     except Exception as e:
90         sg.popup_error_with_traceback("Beim Erzeugen der Excel-Datei trat ein.
↪ Fehler auf:", e)
91
92     window.close()
93
94
95 if __name__ == "__main__":
96     main(applus_configs.serverConfYamlProd)

```

4.7 copy_artikel.py

Beispiel, wie Artikel inklusive Arbeitsplan und Stückliste dupliziert werden kann.

```

1  import pathlib
2  import PyAPplus64
3  import applus_configs
4  import logging
5  import yaml
6  from typing import Optional
7
8
9  def main(confFile: pathlib.Path, artikel: str, artikelNeu: Optional[str] = None) ->
↪ None:
10     # Server verbinden
11     server = PyAPplus64.applus.applusFromConfigFile(confFile)
12
13     # DuplicateBusinessObject für Artikel erstellen
14     dArt = PyAPplus64.duplicate.loadDBDuplicateArtikel(server, artikel)
15
16     # DuplicateBusinessObject zur Demonstration in YAML konvertieren und zurück
17     dArtYaml = yaml.dump(dArt)
18     print(dArtYaml)
19     dArt2 = yaml.load(dArtYaml, Loader=yaml.UnsafeLoader)
20
21     # Neue Artikel-Nummer bestimmen und DuplicateBusinessObject in DB schreiben
22     # Man könnte hier genauso gut einen anderen Server verwenden
23     if (artikelNeu is None):
24         artikelNeu = server.nextNumber("Artikel")
25
26     if not (dArt is None):
27         dArt.setFields({"artikel": artikelNeu})
28         res = dArt.insert(server)
29         print(res)
30
31
32 if __name__ == "__main__":
33     # Logger Einrichten
34     logging.basicConfig(level=logging.INFO)
35     # logger = logging.getLogger("PyAPplus64.applus_db");
36     # logger.setLevel(logging.ERROR)
37
38     main(applus_configs.serverConfYamlTest, "my-artikel", artikelNeu="my-artikel-copy
↪")

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

5.1 Submodules

5.2 PyAPplus64.applus module

```
class PyAPplus64.applus.APplusServer(db_settings: PyAPplus64.applus_db.APplusDBSettings,  
                                     server_settings:  
                                     PyAPplus64.applus_server.APplusAppServerSettings,  
                                     web_settings:  
                                     PyAPplus64.applus_server.APplusWebServerSettings)
```

Bases: object

Verbindung zu einem APplus DB und App Server mit Hilfsfunktionen für den komfortablen Zugriff.

Parameter

- **db_settings** (*APplusDBSettings*) – die Einstellungen für die Verbindung mit der Datenbank
- **server_settings** (*APplusAppServerSettings*) – die Einstellungen für die Verbindung mit dem APplus App Server
- **web_settings** (*APplusWebServerSettings*) – die Einstellungen für die Verbindung mit dem APplus Web Server

completeSQL(*sql*: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], *raw*: bool = False) → str
Vervollständigt das SQL-Statement. Es wird z.B. der Mandant hinzugefügt.

Parameter

- **sql** (*sql_utils.SqlStatement*) – das SQL Statement
- **raw** (*boolean*) – soll completeSQL ausgeführt werden? Falls True, wird die Eingabe zurückgeliefert

Rückgabe das vervollständigte SQL-Statement

Rückgabetyt str

- dbQuery** (*sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], f: Callable[[pyodbc.Row], None], *args: Any, raw: bool = False*) → None
Führt eine SQL Query aus und führt für jede Zeile die übergeben Funktion aus. Das SQL wird zunächst vom Server angepasst, so dass z.B. Mandanteninformation hinzugefügt werden.
- dbQueryAll** (*sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], *args: Any, raw: bool = False, apply: Optional[Callable[[pyodbc.Row], Any]] = None*) → Any
Führt eine SQL Query aus und liefert alle Zeilen zurück. Das SQL wird zunächst vom Server angepasst, so dass z.B. Mandanteninformation hinzugefügt werden.
- dbQuerySingleRow** (*sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], *args: Any, raw: bool = False*) → Optional[pyodbc.Row]
Führt eine SQL Query aus, die maximal eine Zeile zurückliefern soll. Diese Zeile wird geliefert.
- dbQuerySingleRowDict** (*sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], *args: Any, raw: bool = False*) → Optional[Dict[str, Any]]
Führt eine SQL Query aus, die maximal eine Zeile zurückliefern soll. Diese Zeile wird als Dictionary geliefert.
- dbQuerySingleValue** (*sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], *args: Any, raw: bool = False*) → Any
Führt eine SQL Query aus, die maximal einen Wert zurückliefern soll. Dieser Wert oder None wird geliefert.
- dbQuerySingleValues** (*sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], *args: Any, raw: bool = False*) → Sequence[Any]
Führt eine SQL Query aus, die nur eine Spalte zurückliefern soll.

db_conn

Eine pyodbc-Connection zur APplus DB. Diese muss genutzt werden, wenn mehrere Operationen in einer Transaktion genutzt werden sollen. Ansonsten sind die Hilfsmethoden wie *APplusServer.dbQuery()* zu bevorzugen. Diese Connection kann in Verbindung mit den Funktionen aus *PyAPplus64.applus_db* genutzt werden.

db_settings: PyAPplus64.applus_db.APplusDBSettings

Die Einstellungen für die Datenbankverbindung

execUseXMLRowDelete (*table: str, id: int*) → None**getClient** (*package: str, name: str*) → zeep.client.Client

Erzeugt einen zeep - Client. Mittels dieses Clients kann die WSDL Schnittstelle angesprochen werden. Wird als *package* „p2core“ und als *name* „Table“ verwendet und der resultierende client „client“ genannt, dann kann z.B. mittels „client.service.getCompleteSQL(sql)“ vom AppServer ein Vervollständigen des SQLs angefordert werden.

Parameter

- **package** (*string*) – das Packet, z.B. „p2core“
- **name** – der Name im Packet, z.B. „Table“

Rückgabe den Client

Rückgabety Client

getTableFields (*table: str, isComputed: Optional[bool] = None*) → Set[str]

Liefert die Namen aller Felder einer Tabelle.

Parameter

- **table** – Name der Tabelle
- **isComputed** – wenn gesetzt, werden nur die Felder geliefert, die berechnet werden oder nicht berechnet werden

Rückgabe Liste aller Feld-Namen

Rückgabety {str}

getUniqueFieldsOfTable(*table: str*) → Dict[str, List[str]]

Liefert alle Spalten einer Tabelle, die eindeutig sein müssen. Diese werden als Dictionary gruppiert nach Index-Namen geliefert. Jeder Eintrag enthält eine Liste von Feldern, die zusammen eindeutig sein müssen.

isDBTableKnown(*table: str*) → bool

Prüft, ob eine Tabelle im System bekannt ist

makeWebLink(*base: str, **kwargs: Any*) → str

makeWebLinkBauftrag(***kwargs: Any*) → str

makeWebLinkWauftrag(***kwargs: Any*) → str

makeWebLinkWauftragPos(***kwargs: Any*) → str

mkUseXMLRowDelete(*table: str, id: int*) → *PyAPplus64.applus_usexml.UseXmlRowDelete*

mkUseXMLRowInsert(*table: str*) → *PyAPplus64.applus_usexml.UseXmlRowInsert*

Erzeugt ein Objekt zum Einfügen eines neuen DB-Eintrags.

Parameter table (*str*) – DB-Tabelle in die eingefügt werden soll

Rückgabe das XmlRow-Objekt

Rückgabotyp *applus_usexml.UseXmlRowInsert*

mkUseXMLRowInsertOrUpdate(*table: str*) → *PyAPplus64.applus_usexml.UseXmlRowInsertOrUpdate*

Erzeugt ein Objekt zum Einfügen oder Updaten eines DB-Eintrags.

Parameter table (*string*) – DB-Tabelle in die eingefügt werden soll

Rückgabe das XmlRow-Objekt

Rückgabotyp *applus_usexml.UseXmlRowInsertOrUpdate*

mkUseXMLRowUpdate(*table: str, id: int*) → *PyAPplus64.applus_usexml.UseXmlRowUpdate*

nextNumber(*obj: str*) → str

Erstellt eine neue Nummer für das Objekt und legt diese Nummer zurück.

reconnectDB() → None

scripttool: *PyAPplus64.applus_scripttool.APplusScriptTool*

erlaubt den einfachen Zugriff auf Funktionen des ScriptTools

server_conn: *PyAPplus64.applus_server.APplusServerConnection*

erlaubt den Zugriff auf den AppServer

sysconf: *PyAPplus64.applus_sysconf.APplusSysConf*

erlaubt den Zugriff auf die Sysconfig

useXML(*xml: str*) → Any

Ruft `p2core.xml.usexml` auf. Wird meist durch ein UseXMLRow-Objekt aufgerufen.

web_settings: *PyAPplus64.applus_server.APplusWebServerSettings*

Die Einstellungen für die Datenbankverbindung

PyAPplus64.applus.applusFromConfig(*yamlString: str, user: Optional[str] = None, env: Optional[str] = None*) → *PyAPplus64.applus.APplusServer*

Läd Einstellungen aus einer Config-Datei und erzeugt daraus ein APplus-Objekt

PyAPplus64.applus.applusFromConfigDict(*yamlDict: Dict[str, Any], user: Optional[str] = None, env: Optional[str] = None*) → *PyAPplus64.applus.APplusServer*

Läd Einstellungen aus einer Config und erzeugt daraus ein APplus-Objekt

PyAPplus64.applus.applusFromConfigFile(*yamlfile: FileDescriptorOrPath, user: Optional[str] = None, env: Optional[str] = None*) → *PyAPplus64.applus.APplusServer*

Läd Einstellungen aus einer Config-Datei und erzeugt daraus ein APplus-Objekt

5.3 PyAPplus64.applus_db module

class PyAPplus64.applus_db.**APplusDBSettings**(*server: str, database: str, user: str, password: str*)
 Bases: object

Einstellungen, mit welcher DB sich verbunden werden soll.

connect() → pyodbc.Connection
 Stellt eine neue Verbindung her und liefert diese zurück.

getConnectionString() → str
 Liefert den ODBC Connection-String für die Verbindung. :return: den Connection-String

class PyAPplus64.applus_db.**DBTableIDs**
 Bases: object

Klasse, die Mengen von IDs gruppiert nach Tabellen speichert

add(*table: str, *ids: int*) → None
 fügt Eintrag hinzu

Parameter

- **table** (*str*) – die Tabelle
- **id** – die ID

getTable(*table: str*) → Set[int]
 Liefert die Menge der IDs für eine bestimmte Tabelle.

Parameter table (*str*) – die Tabelle

Rückgabe die IDs

PyAPplus64.applus_db.**getUniqueFieldsOfTable**(*cnxn: pyodbc.Connection, table: str*) → Dict[str, List[str]]

Liefert alle Spalten einer Tabelle, die eindeutig sein müssen. Diese werden als Dictionary gruppiert nach Index-Namen geliefert. Jeder Eintrag enthält eine Liste von Feldern, die zusammen eindeutig sein müssen.

PyAPplus64.applus_db.**rawQuery**(*cnxn: pyodbc.Connection, sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], f: Callable[[pyodbc.Row], None], *args: Any*) → None

Führt eine SQL Query direkt aus und führt für jede Zeile die übergeben Funktion aus.

PyAPplus64.applus_db.**rawQueryAll**(*cnxn: pyodbc.Connection, sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], *args: Any, apply: Optional[Callable[[pyodbc.Row], Any]] = None*) → Sequence[Any]

Führt eine SQL Query direkt aus und liefert alle Zeilen zurück. Wenn apply gesetzt ist, wird die Funktion auf jeder Zeile ausgeführt und das Ergebnis ausgegeben, die nicht None sind.

PyAPplus64.applus_db.**rawQuerySingleRow**(*cnxn: pyodbc.Connection, sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], *args: Any*) → Optional[pyodbc.Row]

Führt eine SQL Query direkt aus, die maximal eine Zeile zurückliefern soll. Diese Zeile wird geliefert.

PyAPplus64.applus_db.**rawQuerySingleValue**(*cnxn: pyodbc.Connection, sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], *args: Any*) → Any

Führt eine SQL Query direkt aus, die maximal einen Wert zurückliefern soll. Dieser Wert oder None wird geliefert.

PyAPplus64.applus_db.**row_to_dict**(*row: pyodbc.Row*) → Dict[str, Any]
 Konvertiert eine Zeile in ein Dictionary

5.4 PyAPplus64.applus_scripttool module

class PyAPplus64.applus_scripttool.**APplusScriptTool**(*server: PyAPplus64.applus.APplusServer*)
 Bases: object

Zugriff auf AppServer ScriptTool

Parameter **server** (*APplusServerConnection*) – die Verbindung zum Server

getCurrentDate() → str

getCurrentDateTime() → str

getCurrentTime() → str

getLoginName() → str

getMandant() → str

Liefert den aktuellen Mandanten

getMandantName() → str

Liefert den Namen des aktuellen Mandanten

getSystemName() → str

getUserFullName() → str

getUserName() → str

getXMLDefinition(*obj: str, mandant: str = "", checkFileExists: bool = False*) →
 Optional[lxml.etree.Element]

Läd die XML-Definition als String vom APPServer. und parst das XML in ein minidom-Dokument.

Parameter

- **obj** (*str*) – das Objekt, dessen Definition zu laden ist, „Artikel“ läd z.B. „ArtikelDefinition.xml“
- **mandant** (*str optional*) – der Mandant, dessen XML-Doku geladen werden soll, wenn „“ wird der Standard-Mandant verwendet

Rückgabe das gefundene und mittels ElementTree geparste XML-Dokument

Rückgabotyp ET.Element

getXMLDefinitionObj(*obj: str, mandant: str = ""*) →
 Optional[PyAPplus64.applus_scripttool.XMLDefinition]

Benutzt getXMLDefinitionObj und liefert den Top-Level „Object“ Knoten zurück, falls zusätzlich ein MD5 Knoten existiert, also falls das Dokument wirklich vom Dateisystem geladen werden konnte. Ansonsten wird None geliefert.

Parameter

- **obj** (*str*) – das Objekt, dess Definition zu laden ist, „Artikel“ läd z.B. „ArtikelDefinition.xml“
- **mandant** (*str optional*) – der Mandant, dessen XML-Doku geladen werden soll, wenn „“ wird der Standard-Mandant verwendet

Rückgabe das gefundene und mittels ElementTree geparste XML-Dokument

Rückgabotyp Optional[XMLDefinition]

getXMLDefinitionString(*obj: str, mandant: str = ""*) → str

Läd die XML-Defintion als String vom APPServer. Auch wenn kein XML-Dokument im Dateisystem gefunden wird, wird ein String zurückgeliefert, der einen leeren Top-„Object“ Knoten enthält. Für gefundene XML-Dokumente gibt es zusätzlich einen Top-„MD5“-Knoten.

Parameter

- **obj** (*str*) – das Objekt, dessen Definition zu laden ist, „Artikel“ läd z.B. „ArtikelDefinition.xml“
- **mandant** (*str optional*) – der Mandant, dessen XML-Doku geladen werden soll, wenn „“ wird der Standard-Mandant verwendet

Rückgabe das gefundene XML-Dokument als String

Rückgabetyt str

class PyAPplus64.applus_scripttool.**XMLDefinition**(*root: lxml.etree.Element*)

Bases: object

Repräsentation eines XML-Dokuments

getDuplicate() → Tuple[Set[str], bool]

Extrahiert alle Properties, die für Duplizieren konfiguriert sind. Zudem wird ein Flag geliefert, ob diese Properties ein oder ausgeschlossen werden sollen. :return: Tuple aus allen Properties und ob dies aus- (True) oder ein-(False) zuschließen sind. :rtype: Tuple[Set[str], bool]

root: **lxml.etree.Element**

das Root-Element, repräsentiert „object“ aus Datei.

5.5 PyAPplus64.applus_server module

class PyAPplus64.applus_server.**APplusAppServerSettings**(*appserver: str, appserverPort: int, user: str, env: Optional[str] = None*)

Bases: object

Einstellungen, mit welchem APplus App-Server sich verbunden werden soll.

class PyAPplus64.applus_server.**APplusServerConnection**(*settings: PyAPplus64.applus_server.APplusAppServerSettings*)

Bases: object

Verbindung zu einem APplus APP-Server

Parameter settings (**APplusAppServerSettings**) – die Einstellungen für die Verbindung mit dem APplus Server

getClient (*package: str, name: str*) → zeep.client.Client

Erzeugt einen zeep - Client. Mittels dieses Clients kann die WSDL Schnittstelle angesprochen werden. Wird als *package* „p2core“ und als *name* „Table“ verwendet und der resultierende client „client“ genannt, dann kann z.B. mittels „client.service.getCompleteSQL(sql)“ vom AppServer ein Vervollständigen des SQLs angefordert werden.

Parameter

- **package** (*string*) – das Packet, z.B. „p2core“
- **name** – der Name im Packet, z.B. „Table“

Rückgabe den Client

Rückgabetyt Client

class PyAPplus64.applus_server.**APplusWebServerSettings**(*baseurl: Optional[str] = None*)

Bases: object

Einstellungen, mit welchem APplus Web-Server sich verbunden werden soll.

5.6 PyAPplus64.applus_sysconf module

class PyAPplus64.applus_sysconf.**APplusSysConf**(*server: APplusServer*)

Bases: object

SysConf Zugriff mit Cache über AppServer

Parameter **server** (**APplusServer**) – die Verbindung zum Server

clearCache() → None

getBoolean(*module: str, name: str, useCache: bool = True*) → bool

getDouble(*module: str, name: str, useCache: bool = True*) → float

getInt(*module: str, name: str, useCache: bool = True*) → int

getList(*module: str, name: str, useCache: bool = True, sep: str = ','*) → Optional[Sequence[str]]

getString(*module: str, name: str, useCache: bool = True*) → str

5.7 PyAPplus64.applus_usexml module

class PyAPplus64.applus_usexml.**UseXmlRow**(*applus: APplusServer, table: str, cmd: str*)

Bases: object

Klasse, die eine XML-Datei erzeugen kann, die mittels p2core.useXML genutzt werden kann. Damit ist es möglich APplus BusinessObjekte zu erzeugen, ändern und zu löschen. Im Gegensatz zu direkten DB-Zugriffen, werden diese Anfragen über den APP-Server ausgeführt. Dabei werden die von der Weboberfläche bekannten Checks und Änderungen ausgeführt. Als sehr einfaches Beispiel wird z.B. INSDATE oder UPDDATE automatisch gesetzt. Interessanter sind automatische Änderungen und Checks.

Bei der Benutzung wird zunächst ein Objekt erzeugt, dann evtl. mittels **addField()** Felder hinzugefügt und schließlich mittels **exec()** an den AppServer übergeben. Normalerweise sollte die Klasse nicht direkt, sondern über Unterklassen für das Einfügen, Ändern oder Löschen benutzt werden.

Parameter

- **applus** (**APplusServer**) – Verbindung zu APplus
- **table** (*str*) – die Tabelle
- **cmd** (*str*) – cmd-attribut der row, also ob es sich um ein Update, ein Insert oder ein Delete handelt

addField(*name: str | None, value: Any*) → None

Fügt ein Feld zum Row-Node hinzu.

Parameter

- **name** (*string*) – das Feld
- **value** – Wert des Feldes

addTimestampField(*id: int, ts: Optional[bytes] = None*) → None

Fügt ein Timestamp-Feld hinzu. Wird kein Timestamp übergeben, wird mittels der ID der aktuelle Timestamp aus der DB geladen. Dabei kann ein Fehler auftreten. Ein Timestamp-Feld ist für Updates und das Löschen nötig um sicherzustellen, dass die richtige Version des Objekts geändert oder gelöscht wird. Wird z.B. ein Objekt aus der DB geladen, inspiziert und sollen dann Änderungen gespeichert werden, so sollte der Timestamp des Ladens übergeben werden. So wird sichergestellt, dass nicht ein anderer User zwischenzeitlich Änderungen vornahm. Ist dies der Fall, wird dann bei „exec“ eine Exception geworfen.

Parameter

- **id** (*string*) – DB-id des Objektes dessen Timestamp hinzugefügt werden soll

- **ts** (*bytes*) – Fester Timestamp der verwendet werden soll, wenn None wird der Timestamp aus der DB geladen.

addTimestampIDFields(*id: int, ts: Optional[bytes] = None*) → None

Fügt ein Timestamp-Feld sowie ein Feld *id* hinzu. Wird kein Timestamp übergeben, wird mittels der ID der aktuelle Timestamp aus der DB geladen. Dabei kann ein Fehler auftreten. Intern wird *addTimestampField()* benutzt.

Parameter

- **id** (*string*) – DB-id des Objektes dessen Timestamp hinzugefügt werden soll
- **ts** (*bytes*) – Fester Timestamp der verwendet werden soll, wenn None wird der Timestamp aus der DB geladen.

checkFieldSet(*name: Optional[str]*) → bool

Prüft, ob ein Feld gesetzt wurde

checkFieldsSet(**names: str*) → bool

Prüft, ob alle übergebenen Felder gesetzt sind

exec() → Any

Führt die UseXmlRow mittels useXML aus. Je nach Art der Zeile wird etwas zurückgeliefert oder nicht. In jedem Fall kann eine Exception geworfen werden.

getField(*name: str*) → Any

Liefert den Wert eines gesetzten Feldes

toprettyxml() → str

Gibt das formatierte XML aus. Dieses kann per useXML an den AppServer übergeben werden. Dies wird mittels *exec()* automatisiert.

class PyAPplus64.applus_usexml.**UseXmlRowDelete**(*applus: APplusServer, table: str, id: int, ts: Optional[bytes] = None*)

Bases: *PyAPplus64.applus_usexml.UseXmlRow*

Klasse, die eine XML-Datei für das Löschen eines neuen Datensatzes erzeugen kann. Die Felder *id* und *timestamp* werden automatisch gesetzt. Dies sind die einzigen Felder, die gesetzt werden sollten.

Parameter

- **applus** (*APplusServer*) – Verbindung zu APplus
- **table** (*string*) – die Tabelle
- **id** (*int*) – die zu löschende ID
- **ts** (*bytes optional*) – wenn gesetzt, wird dieser Timestamp verwendet, sonst der aktuelle aus der DB

delete() → None

Führt das delete aus. Evtl. wird dabei eine Exception geworfen. Dies ist eine Umbenennung von *exec()*.

class PyAPplus64.applus_usexml.**UseXmlRowInsert**(*applus: APplusServer, table: str*)

Bases: *PyAPplus64.applus_usexml.UseXmlRow*

Klasse, die eine XML-Datei für das Einfügen eines neuen Datensatzes erzeugen kann.

Parameter

- **applus** (*APplusServer*) – Verbindung zu APplus
- **table** (*string*) – die Tabelle

insert() → int

Führt das insert aus. Entweder wird dabei eine Exception geworfen oder die ID des neuen Eintrags zurückgegeben. Dies ist eine Umbenennung von *exec()*.

class PyAPplus64.applus_usexml.**UseXmlRowInsertOrUpdate**(*applus: APplusServer, table: str*)
 Bases: *PyAPplus64.applus_usexml.UseXmlRow*

Klasse, die eine XML-Datei für das Einfügen oder Ändern eines neuen Datensatzes, erzeugen kann. Die Methode *checkExists* erlaubt es zu prüfen, ob ein Objekt bereits existiert. Dafür werden die gesetzten Felder mit den Feldern aus eindeutigen Indices verglichen. Existiert ein Objekt bereits, wird ein Update ausgeführt, ansonsten ein Insert. Bei Updates werden die Felder *id* und *timestamp* automatisch gesetzt.

Parameter

- **applus** (*APplusServer*) – Verbindung zu APplus
- **table** (*string*) – die Tabelle

checkExists() → int | None

Prüft, ob der Datensatz bereits in der DB existiert. Ist dies der Fall, wird die ID geliefert, sonst None

exec() → int

Führt entweder ein Update oder ein Insert durch. Dies hängt davon ab, ob das Objekt bereits in der DB existiert. In jedem Fall wird die ID des erzeugten oder geänderten Objekts geliefert.

insert() → int

Führt ein Insert aus. Existiert das Objekt bereits, wird eine Exception geworfen.

update(*id: Optional[int] = None, ts: Optional[bytes] = None*) → int

Führt ein Update aus. Falls ID oder Timestamp nicht übergeben werden, wird nach einem passenden Objekt gesucht. Existiert das Objekt nicht, wird eine Exception geworfen.

updateOrInsert() → int

Führt das update oder das insert aus. Evtl. wird dabei eine Exception geworfen. Dies ist eine Umbenennung von *exec*(). Es wird die ID des Eintrages geliefert

class PyAPplus64.applus_usexml.**UseXmlRowUpdate**(*applus: APplusServer, table: str, id: int, ts: Optional[bytes] = None*)

Bases: *PyAPplus64.applus_usexml.UseXmlRow*

Klasse, die eine XML-Datei für das Ändern eines neuen Datensatzes, erzeugen kann. Die Felder *id* und *timestamp* werden automatisch gesetzt.

Parameter

- **applus** (*APplusServer*) – Verbindung zu APplus
- **table** (*string*) – die Tabelle
- **id** (*int*) – die ID des zu ändernden Datensatzes
- **ts** (*bytes optional*) – wenn gesetzt, wird dieser Timestamp verwendet, sonst der aktuelle aus der DB

update() → None

Führt das update aus. Evtl. wird dabei eine Exception geworfen. Dies ist eine Umbenennung von *exec*().

5.8 PyAPplus64.duplicate module

Dupliziert ein oder mehrere APplus Business-Objekte

class PyAPplus64.duplicate.**DuplicateBusinessObject**(*table: str, fields: Dict[str, Any], fieldsNotCopied: Dict[str, Any] = {}, allowUpdate: bool = False*)

Bases: *object*

Klasse, die alle Daten zu einem BusinessObject speichert und zum Duplizieren dieses Objektes dient. Dies beinhaltet Daten zu abhängigen Objekten sowie die Beziehung zu diesen Objekten. Zu einem Artikel wird

z.B. der Arbeitsplan gespeichert, der wiederum Arbeitsplanpositionen enthält. Als Beziehung ist u.a. hinterlegt, dass das Feld „APLAN“ der Arbeitsplans dem Feld „ARTIKEL“ des Artikels entsprechen muss und dass „APLAN“ aus den Positionen, „APLAN“ aus dem APlan entsprechen muss. So kann beim Duplizieren ein anderer Name des Artikels gesetzt werden und automatisch die Felder der abhängigen Objekte angepasst werden. Einige Felder der Beziehung sind dabei statisch, d.h. können direkt aus den zu speichernden Daten abgelesen werden. Andere Felder sind dynamisch, d.h. das Parent-Objekt muss in der DB angelegt werden, damit ein solcher dynamischer Wert erstellt und geladen werden kann. Ein typisches Beispiel für ein dynamisches Feld ist „GUID“.

addDependentBusinessObject (*dObj: Optional[PyAPplus64.duplicate.DuplicateBusinessObject]*,
*args: Tuple[str, str]) → None

Fügt ein neues Unterobjekt zum DuplicateBusinessObject hinzu. Dabei handelt es sich selbst um ein DuplicateBusinessObject, das zusammen mit dem Parent-Objekt dupliziert werden sollen. Zum Beispiel sollen zu einem Auftrag auch die Positionen dupliziert werden. Zusätzlich zum Objekt selbst können mehrere (keine, eine oder viele) Paare von Feldern übergeben werden. Ein Paar („pf“, „sf“) verbindet das Feld „pf“ des Parent-Objekts mit dem Feld „sf“ des Sub-Objekts. So ist es möglich, Werte des Parent-Objekts zu ändern und diese Änderungen für Sub-Objekte zu übernehmen. Üblicherweise muss zum Beispiel die Nummer des Hauptobjekts geändert werden. Die gleiche Änderung ist für alle abhängigen Objekte nötig, damit die neuen Objekte sich auf das Parent-Objekt beziehen.

Parameter

- **dObj** (*DuplicateBusinessObject*) – das Unter-Objekt
- **args** – Liste von Tupeln, die Parent- und Sub-Objekt-Felder miteinander verbinden

allowUpdate

Erlaube Updates statt Fehlern, wenn Objekt schon in DB existiert

dependentObjs: List[Dict[str, Any]]

Abhängige Objekte

fields

die Daten

fieldsNotCopied

Datenfelder, die im Original vorhanden sind, aber nicht kopiert werden sollen

getField(field: str, onlyCopied: bool = False) → Any

Schlägt den Wert eines Feldes nach. Wenn onlyCopied gesetzt ist, werden nur Felder zurückgeliefert, die auch kopiert werden sollen.

insert(server: PyAPplus64.applus.APplusServer) → PyAPplus64.applus_db.DBTableIDs

Fügt alle Objekte zur DB hinzu. Es wird die Menge der IDs der erzeugten Objekte gruppiert nach Tabellen erzeugt. Falls ein Datensatz schon existiert, wird dieser entweder aktualisiert oder eine Fehlermeldung geworfen. Geliefert wird die Menge aller eingefügten Objekte mit ihrer ID.

setFields(upds: Dict[str, Any]) → None

Setzt Felder des DuplicateBusinessObjectes und falls nötig seiner Unterobjekte. So kann zum Beispiel die Nummer vor dem Speichern geändert werden.

Parameter upds – Dictionary mit zu setzenden Werten

table

für welche Tabelle ist das BusinessObject

class PyAPplus64.duplicate.FieldsToCopyForTableCache(server: PyAPplus64.applus.APplusServer)
Bases: object

Cache für welche Felder für welche Tabelle kopiert werden sollen

getFieldsToCopyForTable(table: str) → Set[str]

Bestimmt die für eine Tabelle zu kopierenden Spalten. Dazu wird in den XML-Definitionen geschaut. Ist dort ‚include‘ hinterlegt, werden diese Spalten verwendet. Ansonsten alle nicht generierten Spalten, ohne die ‚exclude‘ Spalten. In jedem Fall werden Spalten wie „ID“, die nie kopiert werden sollten, entfernt.

`PyAPplus64.duplicate.addSachgruppeDependentObjects`(*do*: `PyAPplus64.duplicate.DuplicateBusinessObject`, *server*: `PyAPplus64.applus.APplusServer`, *cache*: *Optional*[`PyAPplus64.duplicate.FieldsToCopyForTableCache`] = *None*) → *None*

Fügt Unterobjekte hinzu, die die Sachgruppenwerte kopieren.

Parameter

- **do** – zu erweiterndes `DuplicateBusinessObject`
- **server** (`APplusServer`) – Verbindung zum APP-Server, benutzt zum Nachschlagen der zu kopierenden Felder
- **cache** (*Optional*[`FieldsToCopyForTableCache`]) – Cache, so dass benötigte Felder nicht immer wieder neu berechnet werden müssen

`PyAPplus64.duplicate.getFieldsToCopyForTable`(*server*: `PyAPplus64.applus.APplusServer`, *table*: *str*, *force*: *bool* = *True*) → *Set*[*str*]

Bestimmt die für eine Tabelle zu kopierenden Spalten. Dazu wird in den XML-Definitionen geschaut. Ist dort ‚include‘ hinterlegt, werden diese Spalten verwendet. Ansonsten alle nicht generierten Spalten, ohne die ‚exclude‘ Spalten. In jedem Fall werden Spalten wie ‚ID‘, die nie kopiert werden sollten, entfernt.

`PyAPplus64.duplicate.initFieldsToCopyForTableCacheIfNeeded`(*server*: `PyAPplus64.applus.APplusServer`, *cache*: *Optional*[`PyAPplus64.duplicate.FieldsToCopyForTableCache`] → `PyAPplus64.duplicate.FieldsToCopyForTableCache`)

Hilfsfunktion, die einen Cache erzeugt, falls dies noch nicht geschehen ist.

`PyAPplus64.duplicate.loadDBDuplicateAPlan`(*server*: `PyAPplus64.applus.APplusServer`, *aplan*: *str*, *cache*: *Optional*[`PyAPplus64.duplicate.FieldsToCopyForTableCache`] = *None*) → *Optional*[`PyAPplus64.duplicate.DuplicateBusinessObject`]

Erstelle `DuplicateBusinessObject` für einzelnen Arbeitsplan.

Parameter

- **server** (`APplusServer`) – Verbindung zum APP-Server, benutzt zum Nachschlagen der zu kopierenden Felder
- **aplan** (*str*) – Aplan, der kopiert werden soll.
- **cache** (*Optional*[`FieldsToCopyForTableCache`]) – Cache, so dass benötigte Felder nicht immer wieder neu berechnet werden müssen

Rückgabe das neue `DuplicateBusinessObject`

Rückgabety `DuplicateBusinessObject`

`PyAPplus64.duplicate.loadDBDuplicateArtikel`(*server*: `PyAPplus64.applus.APplusServer`, *artikel*: *str*, *cache*: *Optional*[`PyAPplus64.duplicate.FieldsToCopyForTableCache`] = *None*, *dupAplan*: *bool* = *True*, *dupStueli*: *bool* = *True*) → *Optional*[`PyAPplus64.duplicate.DuplicateBusinessObject`]

Erstelle `DuplicateBusinessObject` für einzelnen Artikel.

Parameter

- **server** (`APplusServer`) – Verbindung zum APP-Server, benutzt zum Nachschlagen der zu kopierenden Felder
- **artikel** (*str*) – Artikel, der kopiert werden soll

- **cache** (*Optional*[*FieldsToCopyForTableCache*]) – Cache, so dass benötigte Felder nicht immer wieder neu berechnet werden müssen
- **dupAplan** (*bool optional*) – Arbeitsplan duplizieren?
- **dupStueli** (*bool optional*) – Stückliste duplizieren?

Rückgabe das neue DuplicateBusinessObject

Rückgabotyp *DuplicateBusinessObject*

`PyAPplus64.duplicate.loadDBDuplicateBusinessObject` (*server*: `PyAPplus64.applus.APplusServer`, *table*: *str*, *cond*: `PyAPplus64.sql_utils.SqlCondition`, *cache*: *Optional*[`PyAPplus64.duplicate.FieldsToCopyForTableCache`] = *None*, *allowUpdate*: *bool* = *False*) → *Optional*[`PyAPplus64.duplicate.DuplicateBusinessObject`]

Läd ein einzelnes DuplicateBusinessObjekt aus der DB. Die Bedingung sollte dabei einen eindeutigen Datensatz auswählen. Werden mehrere zurückgeliefert, wird ein zufälliger ausgewählt. Wird kein Datensatz gefunden, wird None geliefert.

Parameter

- **server** (`APplusServer`) – Verbindung zum APP-Server, benutzt zum Nachschlagen der zu kopierenden Felder
- **table** (*str*) – Tabelle für das neue DuplicateBusinessObjekt
- **cond** (`sql_utils.SqlCondition`) – SQL-Bedingung zur Auswahl eines Objektes
- **cache** (*Optional*[*FieldsToCopyForTableCache*]) – Cache, so dass benötigte Felder nicht immer wieder neu berechnet werden müssen
- **allowUpdate** (*bool*) – ist Update statt Insert erlaubt?

Rückgabe das neue DuplicateBusinessObject

Rückgabotyp *Optional*[*DuplicateBusinessObject*]

`PyAPplus64.duplicate.loadDBDuplicateBusinessObjectSimpleCond` (*server*: `PyAPplus64.applus.APplusServer`, *table*: *str*, *field*: *str*, *value*: *Optional*[*Union*[*str*, *int*, *float*, `PyAPplus64.sql_utils.SqlParam`, `PyAPplus64.sql_utils.SqlField`, `PyAPplus64.sql_utils.SqlFixed`, `PyAPplus64.sql_utils.SqlDate`, `PyAPplus64.sql_utils.SqlDateTime`, `datetime.datetime`, `datetime.date`, `datetime.time`, *bool*]], *cache*: *Optional*[`PyAPplus64.duplicate.FieldsToCopyForTableCache`] = *None*, *allowUpdate*: *bool* = *False*) → *Optional*[`PyAPplus64.duplicate.DuplicateBusinessObject`]

Wrapper für `loadDBDuplicateBusinessObject`, das eine einfache Bedingung benutzt, bei der ein Feld einen bestimmten Wert haben muss.

Parameter

- **server** (`APplusServer`) – Verbindung zum APP-Server, benutzt zum Nachschlagen der zu kopierenden Felder
- **table** (*str*) – Tabelle für das neue DuplicateBusinessObjekt

- **field** (*str*) – Feld für Bedingung
- **value** – Wert des Feldes für Bedingung
- **cache** (*Optional [FieldsToCopyForTableCache]*) – Cache, so dass benötigte Felder nicht immer wieder neu berechnet werden müssen

Rückgabe das neue DuplicateBusinessObject

Rückgabety *Optional [DuplicateBusinessObject]*

`PyAPplus64.duplicate.loadDBDuplicateBusinessObjects`(*server: PyAPplus64.applus.APplusServer, table: str, cond: PyAPplus64.sql_utils.SqlCondition, cache: Optional [PyAPplus64.duplicate.FieldsToCopyForTableCache] = None, allowUpdate: bool = False*) → *Sequence [PyAPplus64.duplicate.DuplicateBusinessObject]*

Läd eine Liste von DuplicateBusinessObjekten aus der DB. Die Bedingung kann mehrere Datensätze auswählen.

Parameter

- **server** (*APplusServer*) – Verbindung zum APP-Server, benutzt zum Nachschlagen der zu kopierenden Felder
- **table** (*str*) – Tabelle für das neue DuplicateBusinessObjekt
- **cond** (*sql_utils.SqlCondition*) – SQL-Bedingung zur Auswahl eines Objektes
- **cache** (*Optional [FieldsToCopyForTableCache]*) – Cache, so dass benötigte Felder nicht immer wieder neu berechnet werden müssen

Rückgabe Liste der neuen DuplicateBusinessObjects

Rückgabety *Sequence [DuplicateBusinessObject]*

`PyAPplus64.duplicate.loadDBDuplicateBusinessObjectsSimpleCond`(*server: PyAPplus64.applus.APplusServer, table: str, field: str, value: Optional [Union [str, int, float, PyAPplus64.sql_utils.SqlParam, PyAPplus64.sql_utils.SqlField, PyAPplus64.sql_utils.SqlFixed, PyAPplus64.sql_utils.SqlDate, PyAPplus64.sql_utils.SqlDateTime, datetime.datetime, datetime.date, datetime.time, bool]]*, *cache: Optional [PyAPplus64.duplicate.FieldsToCopyForTableCache] = None, allowUpdate: bool = False*) → *Sequence [PyAPplus64.duplicate.DuplicateBusinessObject]*

Wrapper für `loadDBDuplicateBusinessObjects`, das eine einfache Bedingung benutzt, bei der ein Feld einen bestimmten Wert haben muss.

Parameter

- **server** (*APplusServer*) – Verbindung zum APP-Server, benutzt zum Nachschlagen der zu kopierenden Felder
- **table** (*str*) – Tabelle für das neue DuplicateBusinessObjekt
- **field** – Feld für Bedingung

- **value** – Wert des Feldes für Bedingung
- **cache** (*Optional* [*FieldsToCopyForTableCache*]) – Cache, so dass benötigte Felder nicht immer wieder neu berechnet werden müssen

Rückgabe Liste der neuen DuplicateBusinessObjects

Rückgabotyp Sequence [*DuplicateBusinessObject*]

PyAPplus64.duplicate.loadDBDuplicateStueli (*server*: PyAPplus64.applus.APplusServer, *stueli*: str, *cache*: *Optional* [PyAPplus64.duplicate.FieldsToCopyForTableCache] = None) → *Optional* [PyAPplus64.duplicate.DuplicateBusinessObject]

Erstelle DuplicateBusinessObject für einzelne Stückliste.

Parameter

- **server** (*APplusServer*) – Verbindung zum APP-Server, benutzt zum Nachschlagen der zu kopierenden Felder
- **stueli** (*str*) – Stückliste, die kopiert werden soll.
- **cache** (*Optional* [*FieldsToCopyForTableCache*]) – Cache, so dass benötigte Felder nicht immer wieder neu berechnet werden müssen

Rückgabe das neue DuplicateBusinessObject

Rückgabotyp *Optional* [*DuplicateBusinessObject*]

PyAPplus64.duplicate.noCopyFields = {'GUID', 'ID', 'ID_A', 'INSDATE', 'INSUSER', 'MANDANT', 'TIMESTAMP', 'TIMESTAMP_A', 'UPDDATE', 'UPDUSER'}

Menge von Feld-Namen, die nie kopiert werden sollen.

5.9 PyAPplus64.pandas module

Pandas Interface für PyAPplus64.

PyAPplus64.pandas.createSqlAlchemyEngine (*server*: PyAPplus64.applus.APplusServer) → sqlalchemy.engine.base.Engine

Erzeugt eine SqlAlchemy-Engine für die Verbindung zur DB.

PyAPplus64.pandas.exportToExcel (*filename*: Union[str, PathLike[str], pandas._typing.WriteExcelBuffer, pandas.io.excel._base.ExcelWriter], *dfs*: Sequence[Tuple[pandas.core.frame.DataFrame, str]], *addTable*: bool = True) → None

Schreibt eine Menge von Dataframes in eine Excel-Tabelle

Parameter

- **filename** – Name der Excel-Datei
- **dfs** – Liste von Tupeln aus DataFrames und Namen von Sheets.

PyAPplus64.pandas.mkDataFrameColumn (*df*: pandas.core.frame.DataFrame, *makeValue*: Union[Callable, str, List[Union[Callable, str]], Dict[Hashable, Union[Callable, str, List[Union[Callable, str]]]]) → pandas.core.series.Series

Erzeugt für alle Zeilen eines Dataframes eine neuen Wert. Dies wird benutzt, um eine Spalte zu berechnen. Diese kann eine Originalspalte ersetzen, oder neu hinzugefügt werden.

Parameter

- **df** – der Dataframe
- **makeValue** – Funktion, die eine Zeile als Parameter bekommt und den neuen Wert berechnet

`PyAPplus64.pandas.mkHyperlinkDataFrameColumn`(*df: pandas.core.frame.DataFrame, makeOrig: Union[Callable, str, List[Union[Callable, str]], Dict[Hashable, Union[Callable, str], List[Union[Callable, str]]]]], makeLink: Callable[[Any], str]*) → `pandas.core.series.Series`

Erzeugt für alle Zeilen eines Dataframes einen Hyperlink. Dies wird benutzt, um eine Spalte mit einem Hyperlink zu berechnen. Diese kann eine Originalspalte ersetzen, oder neu hinzugefügt werden.

Parameter

- **df** – der Dataframe
- **makeOrig** – Funktion, die eine Zeile als Parameter bekommt und den Wert berechnet, der angezeigt werden soll
- **makeLink** – Funktion, die eine Zeile als Parameter bekommt und den Link berechnet

`PyAPplus64.pandas.pandasReadSql`(*server: PyAPplus64.applus.APplusServer, sql: Union[PyAPplus64.sql_utils.SqlStatementSelect, str], raw: bool = False, engine: Optional[sqlalchemy.engine.base.Engine] = None*) → `pandas.core.frame.DataFrame`

Wrapper für `pd.read_sql` für `sqlalchemy-engine`.

Parameter

- **server** (`APplusServer`) – `APplusServer` für Datenbankverbindung und complete-SQL
- **sql** – das SQL-statement

5.10 PyAPplus64.sql_utils module

Diese Datei enthält Funktionen für den Bau von SQL Statements, besonders SELECT-Statements. Es gibt viel ausgefeiltere Methoden für die Erstellung von SQL Statements. `APplus` benötigt jedoch die Statements als Strings, die dann an `APplus` für Änderungen und erst danach an die DB geschickt werden. Dies erschwert die Nutzung von Tools wie `SqlAlchemy`.

Hier werden einfache Hilfsfunktionen, die auf Strings basieren, zur Verfügung gestellt. `PyODBC` erlaubt Parameter (dargestellt als `,`) in SQL Statements, die dann beim Aufruf gefüllt werden. Dies funktioniert auch im Zusammenspiel mit `APplus`. Oft ist es sinnvoll, solche Parameter zu verwenden.

class `PyAPplus64.sql_utils.SqlCondition`

Bases: `object`

Eine abstrakte Sql-Bedingung. Unterklassen erledigen die eigentliche Arbeit.

getCondition() → `str`

Liefert die Bedingung als String

Rückgabe die Bedingung

Rückgabotyp `str`

class `PyAPplus64.sql_utils.SqlConditionAnd`(*conds: `Union[PyAPplus64.sql_utils.SqlCondition, str]`)

Bases: `PyAPplus64.sql_utils.SqlConditionList`

class `PyAPplus64.sql_utils.SqlConditionBinComp`(*op: str, value1: SqlValue, value2: SqlValue*)

Bases: `PyAPplus64.sql_utils.SqlConditionPrepared`

Bedingung der Form `,value1 op value2'`

Parameter

- **op** (`str`) – der Vergleichsoperator
- **value1** (`SqlValue`) – der Wert, kann unterschiedliche Typen besitzen

- **value2** (*SqlValue*) – der Wert, kann unterschiedliche Typen besitzen

class PyAPplus64.sql_utils.**SqlConditionBool**(*b: bool*)
Bases: *PyAPplus64.sql_utils.SqlConditionPrepared*

Fixe True-oder-False Bedingung

class PyAPplus64.sql_utils.**SqlConditionDateTimeFieldInDay**(*field: str, year: int, month: int, day: int*)

Bases: *PyAPplus64.sql_utils.SqlConditionPrepared*

Liegt Datetime in einem bestimmten Monat?

Parameter

- **field** (*str*) – das Feld
- **year** – das Jahr
- **month** – der Monat
- **day** – der Tag

class PyAPplus64.sql_utils.**SqlConditionDateTimeFieldInMonth**(*field: str, year: int, month: int*)
Bases: *PyAPplus64.sql_utils.SqlConditionPrepared*

Liegt Datetime in einem bestimmten Monat?

Parameter

- **field** (*string*) – das Feld
- **year** – das Jahr
- **month** – der Monat

class PyAPplus64.sql_utils.**SqlConditionDateTimeFieldInRange**(*field: str, datetimeVon: datetime.datetime | None, datetimeBis: datetime.datetime | None*)

Bases: *PyAPplus64.sql_utils.SqlConditionPrepared*

Liegt Datetime in einem bestimmten Zeitraum?

Parameter

- **field** (*str*) – das Feld
- **datetimeVon** – der untere Wert (einschließlich), None erlaubt beliebige Zeiten
- **datetimeBis** – der obere Wert (ausschließlich), None erlaubt beliebige Zeiten

class PyAPplus64.sql_utils.**SqlConditionDateTimeFieldInYear**(*field: str, year: int*)
Bases: *PyAPplus64.sql_utils.SqlConditionPrepared*

Liegt Datetime in einem bestimmten Jahr?

Parameter

- **field** (*str*) – das Feld
- **year** – das Jahr

class PyAPplus64.sql_utils.**SqlConditionEq**(*value1: Optional[Union[SqlValue, bool]], value2: Optional[Union[SqlValue, bool]]*)

Bases: *PyAPplus64.sql_utils.SqlConditionPrepared*

Bedingung der Form ‚v1 is null‘, ‚v2 is null‘, ‚v1 = v2‘, ‚(1=1)‘ oder ‚(0=1)‘

Parameter

- **value1** – der Wert, kann unterschiedliche Typen besitzen
- **value2** – der Wert, kann unterschiedliche Typen besitzen

```

class PyAPplus64.sql_utils.SqlConditionFalse
    Bases: PyAPplus64.sql_utils.SqlConditionPrepared
    False-Bedingung

class PyAPplus64.sql_utils.SqlConditionFieldEq(field: str, value: Optional[Union[SqlValue, bool]])
    Bases: PyAPplus64.sql_utils.SqlConditionEq

class PyAPplus64.sql_utils.SqlConditionFieldGe(field: str, value: SqlValue)
    Bases: PyAPplus64.sql_utils.SqlConditionGe

class PyAPplus64.sql_utils.SqlConditionFieldGt(field: str, value: SqlValue)
    Bases: PyAPplus64.sql_utils.SqlConditionGt

class PyAPplus64.sql_utils.SqlConditionFieldIn
    Bases: PyAPplus64.sql_utils.SqlConditionIn

class PyAPplus64.sql_utils.SqlConditionFieldIsNotNull(field: str)
    Bases: PyAPplus64.sql_utils.SqlConditionIsNotNull

class PyAPplus64.sql_utils.SqlConditionFieldIsNull(field: str)
    Bases: PyAPplus64.sql_utils.SqlConditionIsNull

class PyAPplus64.sql_utils.SqlConditionFieldLe(field: str, value: SqlValue)
    Bases: PyAPplus64.sql_utils.SqlConditionLe

class PyAPplus64.sql_utils.SqlConditionFieldLt(field: str, value: SqlValue)
    Bases: PyAPplus64.sql_utils.SqlConditionLt

class PyAPplus64.sql_utils.SqlConditionFieldStringNotEmpty(field: str)
    Bases: PyAPplus64.sql_utils.SqlConditionPrepared
    Feld soll nicht den leeren String oder null enthalten. Der Ausdruck wird wörtlich übernommen.
    Parameter field (str) – das Feld

class PyAPplus64.sql_utils.SqlConditionGe(value1: SqlValue, value2: SqlValue)
    Bases: PyAPplus64.sql_utils.SqlConditionBinComp
    Bedingung der Form ‚value1 >= value2‘
    Parameter
    • value1 – der Wert, kann unterschiedliche Typen besitzen
    • value2 – der Wert, kann unterschiedliche Typen besitzen

class PyAPplus64.sql_utils.SqlConditionGt(value1: SqlValue, value2: SqlValue)
    Bases: PyAPplus64.sql_utils.SqlConditionBinComp
    Bedingung der Form ‚value1 > value2‘
    Parameter
    • value1 – der Wert, kann unterschiedliche Typen besitzen
    • value2 – der Wert, kann unterschiedliche Typen besitzen

class PyAPplus64.sql_utils.SqlConditionIn
    Bases: PyAPplus64.sql_utils.SqlConditionPrepared
    Bedingung der Form ‚v in ...‘
    Parameter
    • value (SqlValue) – der Wert, kann unterschiedliche Typen besitzen
    • values (Sequence[SqlValue]) – die erlaubten Werte

class PyAPplus64.sql_utils.SqlConditionIsNotNull(v: SqlValue)
    Bases: PyAPplus64.sql_utils.SqlConditionPrepared
    Wert soll nicht null sein

```

Parameter v (SqlValue) – der Wert

class PyAPplus64.sql_utils.**SqlConditionIsNull**(v: SqlValue)
Bases: *PyAPplus64.sql_utils.SqlConditionPrepared*

Wert soll null sein

Parameter v (SqlValue) – das Feld

class PyAPplus64.sql_utils.**SqlConditionLe**(value1: SqlValue, value2: SqlValue)
Bases: *PyAPplus64.sql_utils.SqlConditionBinComp*

Bedingung der Form ‚value1 <= value2‘

Parameter

- **value1** – der Wert, kann unterschiedliche Typen besitzen
- **value2** – der Wert, kann unterschiedliche Typen besitzen

class PyAPplus64.sql_utils.**SqlConditionList**(connector: str, emptyCond: str)
Bases: *PyAPplus64.sql_utils.SqlCondition*

Eine SQL Bedingung, die sich aus einer Liste anderer Bedingungen zusammensetzen. Dies kann eine „AND“ oder eine „OR“ Liste sein.

Parameter

- **connector** (str) – wie werden Listenelemente verbunden (AND oder OR)
- **emptyCond** (str) – Rückgabewert für leere Liste

addCondition(cond: PyAPplus64.sql_utils.SqlCondition | str | None) → None

addConditionEq(value1: Optional[Union[SqlValue, bool]], value2: Optional[Union[SqlValue, bool]])
→ None

addConditionFieldEq(field: str, value: Optional[Union[SqlValue, bool]]) → None

addConditionFieldGe(field: str, value: SqlValue) → None

addConditionFieldGt(field: str, value: SqlValue) → None

addConditionFieldIn()

addConditionFieldIsNotNull(field: str) → None

addConditionFieldIsNull(field: str) → None

addConditionFieldLe(field: str, value: SqlValue) → None

addConditionFieldLt(field: str, value: SqlValue) → None

addConditionFieldStringNotEmpty(field: str) → None

addConditionFieldsEq(field1: str, field2: str) → None

addConditionFieldsGe(field1: str, field2: str) → None

addConditionFieldsGt(field1: str, field2: str) → None

addConditionFieldsLe(field1: str, field2: str) → None

addConditionFieldsLt(field1: str, field2: str) → None

addConditionGe(value1: SqlValue, value2: SqlValue) → None

addConditionGt(value1: SqlValue, value2: SqlValue) → None

addConditionLe(value1: SqlValue, value2: SqlValue) → None

addConditionLt(value1: SqlValue, value2: SqlValue) → None

addConditions(*conds: PyAPplus64.sql_utils.SqlCondition | str | None) → None

getCondition() → str
Liefert die Bedingung als String

Rückgabe die Bedingung

Rückgabotyp str

isEmpty() → bool

class PyAPplus64.sql_utils.**SqlConditionLt**(value1: SqlValue, value2: SqlValue)
Bases: *PyAPplus64.sql_utils.SqlConditionBinComp*

Bedingung der Form ‚value1 < value2‘

Parameter

- **value1** – der Wert, kann unterschiedliche Typen besitzen
- **value2** – der Wert, kann unterschiedliche Typen besitzen

class PyAPplus64.sql_utils.**SqlConditionNot**(cond: PyAPplus64.sql_utils.SqlCondition)
Bases: *PyAPplus64.sql_utils.SqlCondition*

Negation einer anderen Bedingung

Parameter **cond** (*SqlCondition*) – die zu negierende Bedingung

getCondition() → str
Liefert die Bedingung als String

Rückgabe die Bedingung

Rückgabotyp str

class PyAPplus64.sql_utils.**SqlConditionOr**(*conds: Union[PyAPplus64.sql_utils.SqlCondition, str])
Bases: *PyAPplus64.sql_utils.SqlConditionList*

class PyAPplus64.sql_utils.**SqlConditionPrepared**(cond: Union[PyAPplus64.sql_utils.SqlCondition, str])
Bases: *PyAPplus64.sql_utils.SqlCondition*

Bases: *PyAPplus64.sql_utils.SqlCondition*

Eine einfache Sql-Bedingung, die immer einen festen String zurückgibt.

getCondition() → str
Liefert die Bedingung als String

Rückgabe die Bedingung

Rückgabotyp str

class PyAPplus64.sql_utils.**SqlConditionStringStartsWith**(field: str, value: str)
Bases: *PyAPplus64.sql_utils.SqlConditionPrepared*

Feld soll mit einem bestimmten String beginnen

Parameter

- **field** (*str*) – das Feld
- **value** (*str*) – der Wert

class PyAPplus64.sql_utils.**SqlConditionTrue**
Bases: *PyAPplus64.sql_utils.SqlConditionPrepared*

True-Bedingung

class PyAPplus64.sql_utils.**SqlDate**(d: Union[datetime.datetime, datetime.date] =
datetime.datetime(2023, 5, 6, 22, 11, 14, 682145))

Bases: object

Wrapper um DateTime, die die Formatierung erleichtern

Parameter **d** (*Union[datetime.datetime, datetime.date]*) – das Datum

```
class PyAPplus64.sql_utils.SqlDateTime(dt: Union[datetime.datetime, datetime.date] =
                                         datetime.datetime(2023, 5, 6, 22, 11, 14, 682138))
```

Bases: object

Wrapper um DateTime, die die Formatierung erleichtern

Parameter dt (*Union[datetime.datetime, datetime.date]*) – der Zeitpunkt

```
class PyAPplus64.sql_utils.SqlField(fn: str)
```

Bases: object

Wrapper um SQL Feldnamen, die die Formatierung erleichtern

Parameter fn (*str*) – der Feldname

```
class PyAPplus64.sql_utils.SqlFixed(s: str)
```

Bases: object

Wrapper um Strings, die ohne Änderung in SQL übernommen werden

Parameter s (*str*) – der string

```
class PyAPplus64.sql_utils.SqlInnerJoin(table: str, *conds:
                                         Union[PyAPplus64.sql_utils.SqlCondition, str])
```

Bases: *PyAPplus64.sql_utils.SqlJoin*

Ein Inner-Join.

Parameter

- **table** (*str*) – die Tabelle, die gejoint werden soll
- **conds** – Bedingungen, die bereits hinzugefügt werden soll. Weitere können über Attribut *on* hinzugefügt werden.

```
class PyAPplus64.sql_utils.SqlJoin(joinType: str, table: str, *conds:
                                    Union[PyAPplus64.sql_utils.SqlCondition, str])
```

Bases: object

Ein abstrakter Sql-Join

Parameter

- **joinType** (*str*) – Art des Joins, wird in SQL übernommen, z.B. „LEFT JOIN“.
- **table** (*str*) – die Tabelle, die gejoint werden soll
- **conds** – Bedingungen, die bereits hinzugefügt werden soll. Weitere können über Attribut *on* hinzugefügt werden.

getJoin() → str

Liefert den Join als String

on: *PyAPplus64.sql_utils.SqlConditionAnd*

Bedingung des Joins, kann noch nachträglich erweitert werden

```
class PyAPplus64.sql_utils.SqlLeftJoin(table: str, *conds:
                                         Union[PyAPplus64.sql_utils.SqlCondition, str])
```

Bases: *PyAPplus64.sql_utils.SqlJoin*

Ein Left-Join.

Parameter

- **table** (*str*) – die Tabelle, die gejoint werden soll
- **conds** – Bedingungen, die bereits hinzugefügt werden soll. Weitere können über Attribut *on* hinzugefügt werden.

class PyAPplus64.sql_utils.SqlParam

Bases: object

Hilfsklasse, für einen Parameter (?)

class PyAPplus64.sql_utils.SqlStatementSelect(*table: str, *fields: str*)

Bases: object

Klasse, um einfache Select-Statements zu bauen.

Parameter

- **table** (*str*) – die Haupt-Tabelle
- **fields** – kein oder mehrere Felder, die selektiert werden sollen

addFields(**fields: str*) → None

Fügt ein oder mehrere Felder, also auszuwählende Werte zu einem SQL-Statement hinzu.

addFieldsTable(*table: str, *fields: str*) → None

Fügt ein oder mehrere Felder, die zu einer Tabelle gehören zu einem SQL-Statement hinzu. Felder sind Strings. Vor jeden dieser Strings wird die Tabelle mit einem Punkt getrennt gesetzt. Dies kann im Vergleich zu ‚addFields‘ Schreibaarbeit erleichtern.

addGroupBy(**fields: str*) → None

Fügt ein oder mehrere GroupBy Felder zu einem SQL-Statement hinzu.

addInnerJoin(*table: str, *conds: Union[PyAPplus64.sql_utils.SqlCondition, str]*) →

PyAPplus64.sql_utils.SqlInnerJoin

addJoin(*j: PyAPplus64.sql_utils.SqlJoin | str*) → None

Fügt ein Join zum SQL-Statement hinzu. Beispiel: ‚LEFT JOIN personal p ON t.UPDUSER = p.PERSONAL‘

addLeftJoin(*table: str, *conds: Union[PyAPplus64.sql_utils.SqlCondition, str]*) →

PyAPplus64.sql_utils.SqlLeftJoin

fields: List[str]

Liste von auszuwählenden Feldern

getSql() → str

Liefert das SQL-SELECT-Statement als String

groupBy: List[str]

die Bedingung, Default ist True

having: PyAPplus64.sql_utils.SqlConditionList

die Bedingung having, Default ist True

joins: List[PyAPplus64.sql_utils.SqlJoin | str]

Joins mit extra Tabellen

order: Optional[str]

Sortierung

setTop(*t: int*) → None

Wie viele Datensätze sollen maximal zurückgeliefert werden? 0 für alle

table: str

die Tabelle

top: int

wie viele Datensätze auswählen? 0 für alle

where: PyAPplus64.sql_utils.SqlConditionList

die Bedingung, Default ist True

class PyAPplus64.sql_utils.SqlTime(*t: Union[datetime.datetime, datetime.time] =*

datetime.datetime(2023, 5, 6, 22, 11, 14, 682149))

Bases: object

Wrapper um DateTime, die die Formatierung erleichtern

Parameter *t* (*Union[datetime.datetime, datetime.time]*) – die Zeit

PyAPplus64.sql_utils.**SqlValue**

Union-Type aller unterstützter SQL-Werte

alias of *Union[str, int, float, PyAPplus64.sql_utils.SqlParam, PyAPplus64.sql_utils.SqlField, PyAPplus64.sql_utils.SqlFixed, PyAPplus64.sql_utils.SqlDate, PyAPplus64.sql_utils.SqlDateTime, datetime.datetime, datetime.date, datetime.time]*

PyAPplus64.sql_utils.**formatSqlValue**(*v: SqlValue*) → *str*

Formatiert einen Wert für SQL. Je nachdem um welchen Typ es sich handelt, werden andere Formatierungen verwendet.

Parameter *v* (*SqlValue*) – der Wert

Rückgabe der formatierte Wert

Rückgabotyp *str*

PyAPplus64.sql_utils.**formatSqlValueString**(*s: str*) → *str*

Formatiert einen String für ein Sql-Statement. Der String wird in „“ eingeschlossen und Hochkomma im Text maskiert.

Parameter *s* (*str*) – der String

Rückgabe der formatierte String

Rückgabotyp *str*

PyAPplus64.sql_utils.**normaliseDBfield**(*f: str*) → *str*

Normalisiert die Darstellung eines DB-Feldes

PyAPplus64.sql_utils.**normaliseDBfieldList**(*l: Sequence[str]*) → *Sequence[str]*

Normalisiert eine Menge von DB-Feldern

PyAPplus64.sql_utils.**normaliseDBfieldSet**(*s: Set[str]*) → *Set[str]*

Normalisiert eine Menge von DB-Feldern

PyAPplus64.sql_utils.**sqlParam** = <PyAPplus64.sql_utils.SqlParam object>

Da SqlParam keinen Zustand hat, reicht ein einzelner statischer Wert

5.11 PyAPplus64.utils module

PyAPplus64.utils.**checkDirExists**(*dir: Union[str, pathlib.Path]*) → *pathlib.Path*

Prüft, ob ein Verzeichnis existiert. Ist dies nicht möglich, wird eine Exception geworfen.

Parameter *dir* (*Union[str, pathlib.Path]*) – das Verzeichnis

Rückgabe den normalisierten Pfad

Rückgabotyp *pathlib.Path*

PyAPplus64.utils.**containsOnlyAllowedChars**(*charset: Set[str], s: str*) → *bool*

Enthält ein String nur erlaubte Zeichen?

PyAPplus64.utils.**formatDateTimeForAPplus**(*v: Union[datetime.datetime, datetime.date, datetime.time]*) → *str*

Formatiert ein Datum oder eine Uhrzeit für APplus

5.12 Module contents

p

PyAPplus64, 43
PyAPplus64.applus, 21
PyAPplus64.applus_db, 24
PyAPplus64.applus_scripttool, 25
PyAPplus64.applus_server, 26
PyAPplus64.applus_sysconf, 27
PyAPplus64.applus_usexml, 27
PyAPplus64.duplicate, 29
PyAPplus64.pandas, 34
PyAPplus64.sql_utils, 35
PyAPplus64.utils, 42