



Geoprocessing mit python, Nutzung am Geodatenzentrum

Kerstin Reinhold

Bundesamt für Kartographie und Geodäsie



- Vorstellung Geodatenzentrum
- unser Weg zu python
- Pluspunkte für python
- Verwendung von python-Modulen
- Programm-Beispiele:
 1. Aufbereitung einer Geodatabase zur Prüfung der Ländergrenzen im Basis-DLM
 2. Tägliches Update von Metadaten-Shapefiles
 3. Datenaktualisierung für Webdienst



Das Geodatenzentrum am BKG

Zentrale Bereitstellung klein- und mittelmaßstäbiger digitaler Kartendaten für die Bundesverwaltung und Verkauf der Daten und Dienste an übrige Nutzer.

Arbeitsaufgaben:

- Auftragsbearbeitung
- Datenupdates
- Datenprüfung
- Metadatenpflege
- Webservices



Startseite | Seitenverzeichnis | Impressum | Kontakt | English

Bundesamt für
Kartographie und Geodäsie

GeoDatenZentrum **BKG-Homepage**

Suche

Aktuelles
Auskunft über Daten & Dienste
Web-Anwendungen
Online-Shop
Download
Infos und Hinweise
Über uns
Links
GDZ-Intern

Informationsmaterial
Glossar
Seite drucken

Startseite

GeoDatenZentrum

Zentraler Geodatenservice für amtliche Geobasisdaten

Hier erhalten Sie vielfältige Informationen über die Geobasisdaten der Bundesländer und des Bundes. Nutzen Sie unsere Dienste und interaktiven Karten für Bestellung, Download, Suche oder Verarbeitung von Geoinformationen.

Topthema: DOP-Viewer (Luftbilder Deutschlands)

Entdecken Sie Deutschland aus der Vogelperspektive! Erkunden Sie Orte, Landschaften oder Ihr Zuhause mit den amtlichen Digitalen Orthophotos (DOP) der Vermessungsverwaltungen aller Bundesländer.

[Zum DOP-Viewer](#)

MetaDaten
Geodaten
SHOP Dienste
SHOP Karten

Login - autorisierte Nutzung

Aktuelle Meldungen

- Online-Koordinatentransformation erneuert
- Web Map Service DLM1000
- ATKIS Basis-DLM Vertriebsdaten aktualisiert
- Verwaltungsgebiete VG250 und VG1000 aktualisiert verfügbar (ohne Einwohnerzahlen)
- Verwaltungsgrenzen unter neuem Namen und in neuer Produktstruktur
- Umbenennung der Digitalen Geländemodelle
- Digitale Orthophotos mit einer Auflösung 20 cm
- DLM1000 in überarbeiteter Fassung
- Geographische Namen GN250 in neuer Struktur mit Stand 31.12.2008 verfügbar
- DLM250 in aktualisierter Version verfügbar

Weitere WEB - Services

- Koordinatentransformation
- Suche geografischer Namen
- Suche historischer Namen





Unser Weg zu python

- Beginn der Arbeiten mit Arc/Info-Workstation auf Unix-Servern mit zentraler Datenhaltung und Arbeitsumgebung, Datenverarbeitung im Batchbetrieb per AML-Script.
- Umstieg auf ArcGIS-Desktop → Rückschritt in Hinblick auf zentrale Datenhaltung, einheitliche Arbeitsumgebung und Verarbeitung großer Datenmengen.
- Erste Lösungen mittels VbA-Programmierung → Tools waren immer an die Desktop-Anwendung gebunden.
- Auch Java-Programmierung wurde ausprobiert.
- Weiterentwicklung von ArcGIS hin zu Batchfähigkeit durch python-scripting ließ die Hoffnung aufkeimen, automatisierte Arbeitsprozesse mit ArcGIS realisieren zu können.





Pluspunkte für python



- + Leicht zugängliche Script-Sprache, aber auch objektorientierte Programmierung möglich
- + schnelles Ausprobieren einzelner Befehle aus der Entwicklungsumgebung heraus (IDLE - Python Shell)
- + Einfache Scripte für schnell zu lösende Probleme, z.B. Dateien umbenennen, Verzeichnisse/Dateien durchsuchen
- + Objektorientierte Programmierung möglich, Erstellen wiederverwendbarer Klassen (z.B. Logger, MeineZeit, ...)
- + Entwicklungsumgebung IDLE ist im Lieferumfang enthalten, auch Eclipse kann für Python aufgerüstet werden (PyDev-Modul)



Pluspunkte für python

- + Kostenloser Download von neuen Versionen und Libraries
- + Eine Vielzahl von Libraries existiert für die verschiedensten Anwendungsbereiche
- + Ausführliches Tutorial im Internet
- + plattformunabhängig
- + Auch in FME-Workbenches verwendbar (pythoncaller, Startup- und Shutdown-Script)





Häufig verwendete Libraries

- | | |
|---------------------|---|
| os, shutil, zipfile | Werkzeug für Datenhaltung, Dateiverwaltung, Datensicherung auf Systemebene |
| dom | Lesen/Schreiben von XML-Strukturen |
| cx_oracle | Datenbankabfragen |
| email, smtplib | E-Mail-Meldungen abfeuern |
| arcgisscripting | Arbeit mit Shapefiles oder Geodatabases
(arcgisscripting ist keine echte Library
→ Modul arcpy in ArcGIS10) |





Allgemeine Regeln für Programme

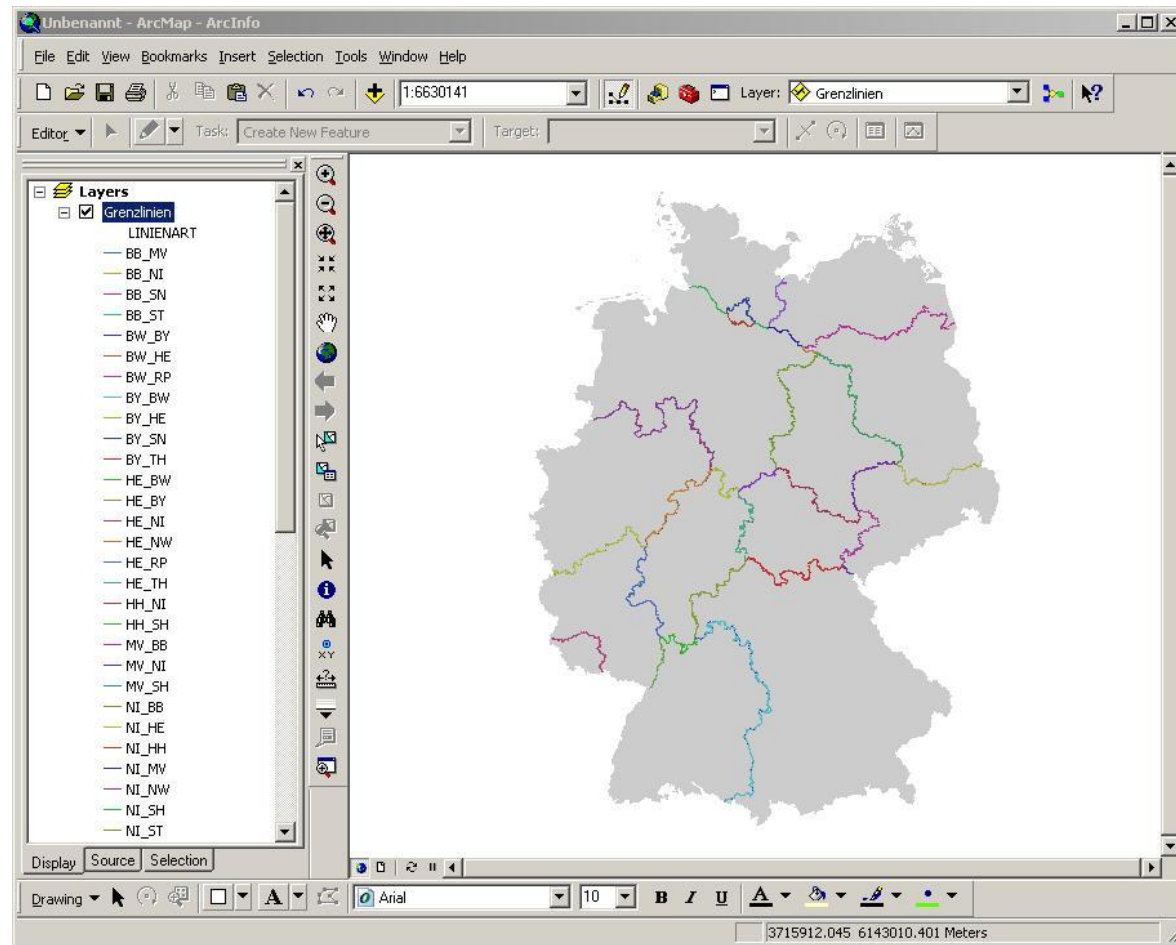
- Batchfähigkeit = Aufruf über Kommandozeile ohne graphische Oberfläche möglich
- Aufruf ohne Parameter liefert immer Syntax und Kurzbeschreibung
- Zentrale Ablage der Programme auf dem Fileserver
- Modularität = für jeden Arbeitsschritt ein Programm
- Fortschrittsanzeigen in Form von verständlichen Ausschriften
- Logging

Beispiel 1 - Aufbereitung einer Geodatabase zur Prüfung der Ländergrenzen im Basis-DLM

Aufgabe:

Prüfung der Passfähigkeit der Linien der Bundeslandgrenzen auf gemeinsamen Grenzabschnitten.

- Originaldaten dürfen nicht verändert werden.
- 54 Grenzabschnitte sind zu prüfen.





Beispiel 1 - Aufbereitung einer Geodatabase zur Prüfung der Ländergrenzen im Basis-DLM

```
c:\ Eingabeaufforderung
E:\>E:\bdlm_grenzanpassung\Tools\GrenzLinien.py
Grenzpruefung
Aufruf: GrenzLinien <option>
options:
  del          - Loescht alle vorhandenen Grenzlinien
  add [<land>] - Fuegt Grenzlinien ein, wahlweise eines oder aller Laender
  set         - Setzt den Subtype Linienart (Voraussetzung fuer Topologie)
  tpl        - definiert die Grenz-Topologie
  def        - definiert Subtypes fuer Werte im Feld LINIENART
E:\>
```

Arbeitsschritte:

- Kopieren der Ländergrenzen in eigenständige Geodatabase
- Definition von Subtypes zur Klassifizierung der Grenzabschnitte
- Grenzabschnitte attributieren
- Topologie-Layer anlegen
- Topologie-Regeln definieren



Beispiel 1 - Aufbereitung einer Geodatabase zur Prüfung der Ländergrenzen im Basis-DLM

Grenzabschnitte attributieren:

Vierstelligen Code für Subtype Linienart vergeben

z.B. Brandenb. Grenze zu Mecklenburg-Vorpommern

„BB_MV“ = 1213

Mecklenb. Grenze zu Brandenburg

„MV_BB“ = 1312





Beispiel 1 - Aufbereitung einer Geodatabase zur Prüfung der Ländergrenzen im Basis-DLM

```
laender = ['bb', 'bw', 'by', 'he', 'hh', 'mv', 'ni', 'nw', 'rp', 'sh', 'sl', 'sn', 'st', 'th']
codes = {'bb':12, 'bw':98, 'by':99, 'he':96, 'hh':92, 'mv':13, 'ni':93, 'nw':95, 'rp':97, 'sh':91, 'sl':10, 'sn':14, 'st':15, 'th':16}

for land in laender:
    code1 = codes[land] * 100

    fLayer = land + 'Lyr'
    query = "LOWER(\ 'LAND\ ') = '" + land + "'"
    if gp.Exists(fLayer):
        gp.Delete(fLayer)
    gp.MakeFeatureLayer(grenzlinien, fLayer, query)

for land2 in laender:
    if land <> land2:
        code = code1 + codes[land2]

        #Nachbarlaender in Pufferzone suchen
        puffer = "E:\bdlm_grenzanpassung\Daten\GrzPruefung.gdb\Puffer_Landesgrenze_" + land2
        gp.SelectLayerByLocation(fLayer, "intersect", puffer)
        rows = gp.UpdateCursor(fLayer)
        row = rows.Next()

        #Grenzabschnitte attributieren mit Code fuer land+'_'+land2
        x = 0
        while row:
            x += 1
            row.setValue('Linienart', code)
            rows.UpdateRow(row)
            row = rows.Next()
        if x > 0:
            print 'Grenzobjekte zugewiesen: '+land+' - '+land2+' ('+str(code)+'), '+str(x)+' Objekte'
```



Beispiel 1 - Aufbereitung einer Geodatabase zur Prüfung der Ländergrenzen im Basis-DLM

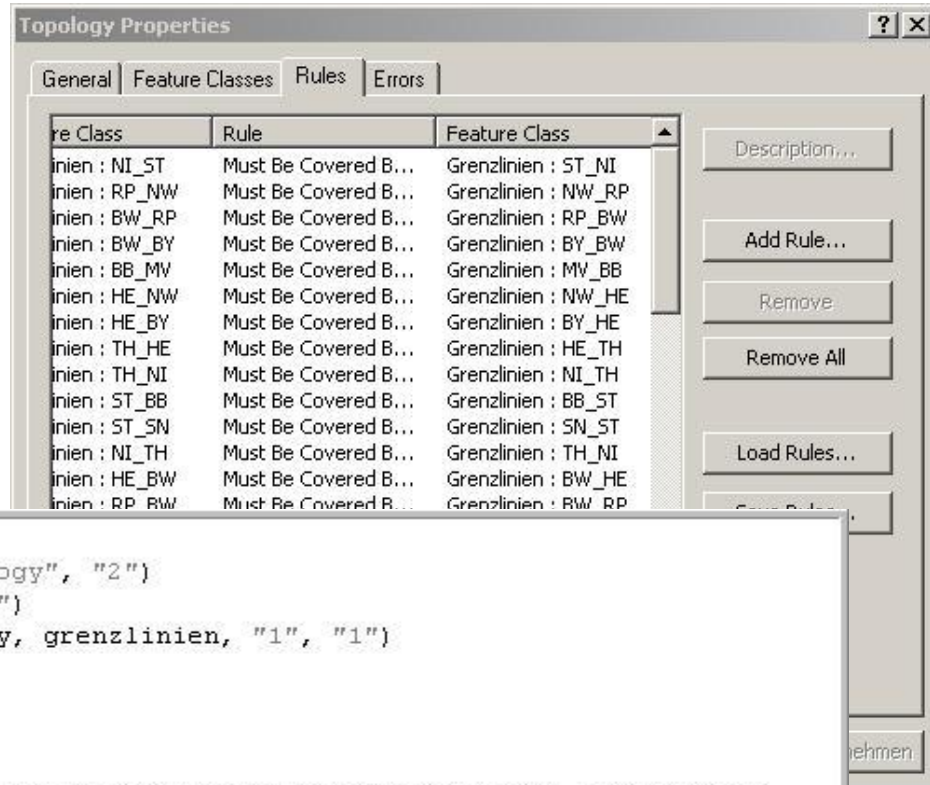
Topologie-Regeln definieren:

```
#alle vorkommenden Linienarten je Land ermitteln
#und entsprechende Topology-Rule daraus ableiten
#z.B.: Linienart SN_BB = 1412
# --> 1412 muss auf 1214 liegen!
for land in laender:
    code1 = codes[land] * 100
    linienArts = []
    query = "Linienart > "+str(code1)+" and Linienart < "+str(code1 + 100)
    rows = gp.SearchCursor(grenzlinien,query)
    row = rows.Next()
    while row:
        linienArts.append(row.getValue('linienart'))
        row = rows.Next()

#fuer Rules zusammengehörige Linienarten ermitteln
#hierfür brauchen wir die Linienart als Value, nicht als Code!
for la in linienArts:
    txtL1 = ''
    txtL2 = ''
    l1 = str(la)[:2]
    l2 = str(la)[-2:]
    for land in codes.keys():
        if codes[land] == int(l1):
            txtL1 = land.upper()
        if codes[land] == int(l2):
            txtL2 = land.upper()
    rules[txtL1+'_'+txtL2] = txtL2+'_'+txtL1
```

Beispiel 1 - Aufbereitung einer Geodatabase zur Prüfung der Ländergrenzen im Basis-DLM

**Topologie-Layer anlegen
und
Regeln hinzufügen:**



```
#Topology anlegen
gp.CreateTopology_management(basisDLM, "GrenzTopology", "2")
grzTopology = os.path.join(basisDLM, "GrenzTopology")
gp.AddFeatureClassToTopology_management(grzTopology, grenzlinien, "1", "1")

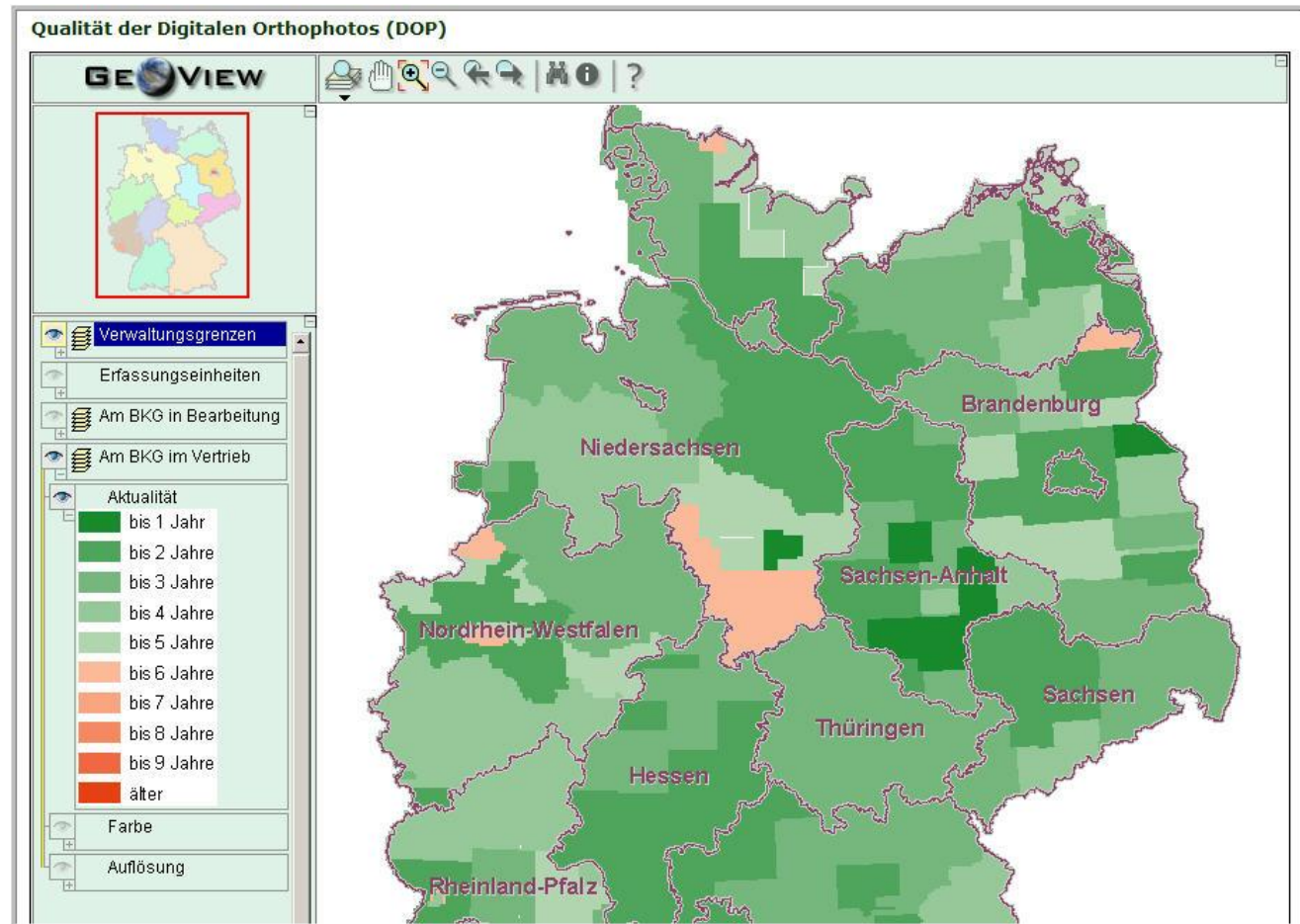
#Regeln hinzufügen
for r1 in rules.keys():
    r2 = rules[r1]
    gp.AddRuleToTopology_management(grzTopology, "Must Be Covered By Feature Class Of (Line-Line)",
                                    grenzlinien, r1, grenzlinien, r2)
```

Beispiel 2 - Tägliches Update von Metadaten-Shapefiles

Aufgabe

Tägliche Aktualisierung der Datengrundlage für den Metadaten-Webdienst

- Datenbank-übergreifende SDE-Views verwenden
- über Scheduler abfeuern





Beispiel 2 - Tägliches Update von Metadaten-Shapefiles

```
C:\ Command Prompt  
E:\Tools>createDOPMetaShapes.py  
Orthophoto Services - Aktualisieren der Metadaten-Shapefiles  
Aufruf: python createDOPMetaShape [U!L!UL]
```

Arbeitsschritte

- Exportieren von SDE-Views in Shapefiles
- Fieldmapping durchführen
- alte Shapefiles gegen neue austauschen

!!! WICHTIG: Logging von Erfolg/Misserfolg



Beispiel 2 - Tägliches Update von Metadaten-Shapefiles

Fieldmapping

```
#-----  
#FieldMappings  
fldMaps = gp.createObject("FieldMappings")  
fldMaps.AddTable(os.path.join(connection,fClass))  
  
#Felder SE_ANNO_CAD_DATA, AREA, LEN nicht mappen!  
rm1 = fldMaps.FindFieldMapIndex("SE_ANNO_CAD_DATA")  
if rm1 <> -1:  
    fldMaps.RemoveFieldMap(rm1)  
rm2 = fldMaps.FindFieldMapIndex("AREA")  
if rm2 <> -1:  
    fldMaps.RemoveFieldMap(rm2)  
rm3 = fldMaps.FindFieldMapIndex("LEN")  
if rm3 <> -1:  
    fldMaps.RemoveFieldMap(rm3)  
rm = fldMaps.FindFieldMapIndex("ZKZ")  
if rm <> -1:  
    fldMaps.RemoveFieldMap(rm)  
rm = fldMaps.FindFieldMapIndex("METADATEN")  
if rm <> -1:  
    fldMaps.RemoveFieldMap(rm)  
rm = fldMaps.FindFieldMapIndex("JAHR_STAND")  
if rm <> -1:  
    fldMaps.RemoveFieldMap(rm)  
#Datum_Lief soll in Lieferung umbenannt werden  
rm1 = fldMaps.FindFieldMapIndex("DATUM_LIEF")  
if rm1 <> -1:  
    fm = fldMaps.GetFieldMap(rm1)  
    fm.OutputField.Name = "LIEFERUNG"
```



Beispiel 2 - Tägliches Update von Metadaten-Shapefiles

neues Shapefile erzeugen

```
#Featureclass nach Shapefile kopieren
gp.workspace = dirShape
fileAlt = fileName + '_' + heute
try:
    # bisheriges Shapefile umbenennen
    ...

    gp.scheitert hier am Exklusiv Lock des laufenden Webservice
    if gp.exists(shpFile):
        gp.rename(shpFile,shpAlt)
    ...

    for dat in os.listdir(dirShape):
        if os.path.splitext(dat)[0] == fileName:
            neu = dat.replace(fileName,fileAlt)
            shutil.move(os.path.join(dirShape,dat),os.path.join(dirShape,neu))

    #neues Shapefile aus DB exportieren
    gp.FeatureClassToFeatureClass_conversion(os.path.join(connection,fClass), dirShape, shpFile, "", fldMaps, "")
    log(logFile,jetzt+" - INFO: Shapefile "+shpFile+" wurde aktualisiert.\n")
except:
    log(logFile,jetzt+' - ERROR: Shapefile '+shpFile+' konnte nicht erzeugt werden!\n')
    #print 'ERROR: Shapefile '+shpFile+' konnte nicht erzeugt werden!'
    if gp.exists(shpAlt):
        gp.rename(shpAlt,shpFile)

if gp.exists(shpAlt):
    gp.delete_management(shpAlt)

del gp
return
```



Beispiel 3 - Datenaktualisierung für Webdienst

DOP-VIEWER
www.geodatenzentrum.de/dienste/dop_viewer

The screenshot shows the DOP-Viewer interface. The main map area displays an aerial view of Leipzig, Germany, with several streets highlighted in orange and yellow. The highlighted streets include Gerberstraße, Tröndlinring, and Willy-Brandt-Platz. The map also shows administrative boundaries (Verwaltungseinheiten) and digital orthophotos. The interface includes a toolbar at the top with various navigation and tool icons, a legend on the left side, and a status bar at the bottom showing the UTM32 coordinate system and a scale bar.

Legend:

- Verwaltungseinheiten
- Basis-DLM-Overlay
- Digitale Orthophotos

Status Bar: UTM32 X= 735.041 Y= 5.693.410 0 90 m



Beispiel 3 - Datenaktualisierung für Webdienst

Aufgabe

Aufbereitung ausgewählter Objektbereiche (Ortslage, Verkehr) des Basis-DLM für einen Webservice zwecks Überlagerung des Luftbildservices

- selektieren, zusammenfassen, generalisieren von Features
- Nutzung von FME-Technologie
- Aktualisierung ca. vierteljährlich über FME-Server

Beispiel 3 - Datenaktualisierung für Webdienst

Problem bei Straßen mit getrennten Fahrbahnen:

- keine Straßennamen an den Fahrbahnen verfügbar
- Verknüpfung von Fahrbahn mit Komplexobjekt notwendig
- Relationen mit FME nur sehr zeit- und ressourcenaufwändig realisierbar





Beispiel 3 - Datenaktualisierung für Webservice

Lösung:

Einsatz des gp-Tools *AddJoin* im Shutdown-Script der FME-Workbench

```
Shutdown Python Script
36
37 #Join zwischen Fahrbahnen (ver01_l) und komplexer Strasse (ver01_k)
38 gp.SelectLayerByAttribute_management(ver01_lyr, "NEW_SELECTION", "\"KO\" <> '0
39 gp.AddJoin_management(ver01_lyr, "KO", ver01_k, "OBJNR", "KEEP_COMMON")
40 logger.log('Join zwischen ver01_l und ver01_k angelegt',1)
41
42 # Namensfelder GN und KN uebernehmen
43 gp.CalculateField_management(ver01_lyr, ver01_l+".GN", "["+ver01_k+".GN]", "VF
44 logger.log('Feld GN uebernommen',1)
45 gp.CalculateField_management(ver01_lyr, ver01_l+".KN", "["+ver01_k+".KN]", "VF
46 logger.log('Feld KN uebernommen',1)
47
48 #diesen Join entfernen
49 gp.RemoveJoin(ver01_lyr,ver01_k)
50
Options
OK Abbrechen
```



Vielen Dank für Ihre Aufmerksamkeit

