BfG-2011

Report

Digital elevation models of German waterway and navigation authorities

Version: **0.1.0**

26.06.2020

SAP number: M39630304065

Number of pages: 43

Compiled by: Arnd Weber
Department U3
Vegetation Studies, Landscape Management
Federal Institute of Hydrology
Koblenz, Germany

Citation: Weber, A. (2020): Digital elevation models of German waterway and navigation authorities - Version 0.1.0. Bundesanstalt für Gewässerkunde, Koblenz, Germany. BfG-Bericht 2011. 43 p. DOI: 10.5675/BfG-2011

This document provides the code to process digital elevation models of the German waterway and navigation authorities (Wasserstraßen- und Schifffahrtsverwaltung des Bundes, WSV) while being inside the local network of the Federal Institute of Hydrology (Bundesanstalt für Gewässerkunde, BfG). Locally stored data are aggregated, retiled and finally exported into machine-readable formats to be published as open data datasets:

Weber, A. (2020): Digital elevation model (DEM 1) of the River Elbe floodplain between Schmilka and Geesthacht, Germany. PANGAEA. https://doi.org/10.1594/PANGAEA.919293

Weber, A. (2020): Digital elevation model (DEM 1) of the River Rhine floodplain between Iffezheim and Kleve, Germany. PANGAEA. https://doi.org/10.1594/PANGAEA.919308

Contents

1 Rationale

Over the last two decades numerous digital elevation models of waterways (Digitales Geländemodell des Wasserlaufes, DGM-W) with high spatial resolution have been produced for the WSV under supervision of the federal institute of hydrology (Bundesanstalt für Gewässerkunde, BfG). These datasets were published by the WSV as open data in 2016 through the official, governmental open data platform govdata.de (Wasserstraßen- und Schifffahrtsverwaltung des Bundes (WSV), 2016):

> https://www.govdata.de/web/guest/suchen/-/details/digitales-gelandemodell-des-wasserlaufs-dgm-w

and are available for download from:

> https://atlas.wsv.bund.de/inspire/atom/hoehen.xml

Unfortunately, data were provided in different data structures, different (proprietary) formats, partially they were even zipped and no presistent identifier or url exists causing major difficulties to dynamically reuse these data products.

To overcome these problems, we decided to reaggregate the original raw data, harmonize the tiling and republish the data as open, machine-readable datasets with persistent identifiers through a scientific data repository:

> https://www.pangaea.de

The digital elevation model for selected floodplains are now available as:

Weber, A. (2020). Digital elevation model (DEM 1) of the River Elbe floodplain between Schmilka and Geesthacht, Germany. PANGAEA. https://doi.org/10.1594/PANGAEA.919293

Weber, A. (2020). Digital elevation model (DEM 1) of the River Rhine floodplain between Iffezheim and Kleve, Germany. PANGAEA. https://doi.org/10.1594/PANGAEA.919308

2 System requirements

To completely rerun this code a number of requirements need to be fulfilled:

- a PC with Windows OS
- a recent installation of R (R Core Team, 2019) extended by additional packages:
  - knitr (Xie, 2014, Xie (2015), Xie (2020))
  - reticulate (Ushey, Allaire, & Tang, 2019)
  - stringr (Wickham, 2019)
  - rgdal (Bivand, Keitt, & Rowlingson, 2019)
  - rgeos (Bivand & Rundel, 2019)
  - leaflet (Cheng, Karambelkar, & Xie, 2019)
  - rosm (Dunnington, 2019)
  - prettymapr (Dunnington, 2017)
  - pangaear (Chamberlain, Woo, MacDonald, Zimmerman, & Simpson, 2020)
- a recent installation of ArcGIS Desktop (ESRI, 2018)
- a recent installation of Python (usually delivered by ArcGIS Desktop) (Rossum, 1995, Python Software Foundation (2017))
- a recent installation of GDAL (GDAL/OGR contributors, 2019)
- read access to the BfG's file system for the original data
- read and write access to the BfG's ArcGIS Enterprise Geodatabase

3 Processing

Since BfG's Department M5 was involved in the technical development and quality assurance of the digital elevation models, copies of the originally by the contractors delivered datasets exist in the BfG's local file system.

These datasets were converted into the published format in four steps:

1. regularly tiled (1 km²), elevation datasets (Arc/Info ASCII Grid) were aggregated to mosaic datasets and stored in an ArcGIS Enterprise Geodatabase
2. a new tiling - based on gauging stations and memory size optimizations - was generated
3. the mosaic datasets were clipped based on the new tiling and exported as Arc/Info ASCII Grid
4. the Arc/Info ASCII Grid datasets were converted and compressed to GeoTIFF

4 Code

4.1 Import of the original data and merging it into a mosaic in an Enterprise Geodatabase

4.1.1 River Elbe

The digital elevation model for River Elbe covers the active floodplain of River Elbe between the Czech-German border near Schmilka and the weir in Geesthacht with 1 m spatial resolution. The dataset was generated through aerial laser scanning (ALS) for terrestrial parts of the floodplain between April 2003 and December 2006 and echo sounding for aquatic parts of the central water course by the local waterway and navigation authorities (WSV) throughout the year 2006 (Brockmann, Großkordt, & Schumann, 2008a). Parts not covered by any of the two data collection methods were filled through linear interpolation (Brockmann et al., 2008a). A small section of the model was updated later to incorporate the dike relocation area Lenzen which became connected to the floodplain in 2011 (Brockmann & Schumann, 2012), so that the resulting full mosaic dataset describes the topographic state of 2011.

```
#!/opt/i4/python-2.7.17/bin/python
# -*- coding: utf-8 -*-
#########
import os
import sys
import shutil

try:
    import arcpy
except ImportError as error:
    print "ArcGIS mit dem Modul {0} muss installiert sein.".format(error.message[16:])
    print "Zur vollstaendigen Installation von ArcGIS finden Sie hier eine Beschreibung:"
    print "http://voss-wiki.bafg.de/instanzen/giswiki/doku.php?id=arcgis_106"

##########
# load the i4 module
try:
    sys.path.append(r"\\mt2.fs.bafg.de\Software\Allgemein\GIS\Inform_4_fuer_ArcGIS_10" + \
                    r".x-Entwicklerversion\inform")
    import i4common
    import xyz2asc
except ImportError as error:
    print r"Das Modul {0} kann nicht geladen werden.".format(error.message[16:])
    print r"Stellen Sie bitte sicher, dass das Netzlaufwerk '\\mt2.fs.bafg.de\Fachdat" + \
          r"en' (Z:) von Ihrem PC erreichbar ist."
    sys.exit(1)
```

```
##########
# Define variables
# source_paths
source_path_dgm001 = r"\\mt2.fs.bafg.de\Fachdaten\Produkte\Befliegungen_M53\_Elbe\200" + \
                     r"3-2006_FE-Datenauswertung-Elbe_km0-586\!FE-Datenauswertung_Elb" + \
                     r"e_Los1-3!"
source_path_dgm001_2011 = r"\\mt2.fs.bafg.de\Fachdaten\Produkte\Befliegungen_M53\_Elb" + \
                          r"e\2003-2011_Elbe-Lenzen_km476-485\etrs89_grs80_utm33-7+dh" + \
                          r"hn92\01_dgmw\01m\02_esri-asciigrid\01_einzeln"

# dest_path for temporary raster data
dest_path = r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBAS" + \
            r"IS\GEODATEN\TOPOGRAFIE\HOEHENMODELL"

# gdb_path for the connection data to the Enterprise GDB
gdb_path = i4common.eGDB(readonly=False)
fgdb_path = r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBAS" + \
            r"IS\GEODATEN\TOPOGRAFIE\HOEHENMODELL\DGM.gdb"

##########
# PREPARATION
##
# Define the coordinate reference systems
ETRS_1989_UTM_Zone_33N = arcpy.SpatialReference("ETRS 1989 UTM Zone 33N")
ETRS_1989_UTM_Zone_33N_7stellen = arcpy.SpatialReference(r"\\mt2.fs.bafg.de\Fachdaten" + \
    r"\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS\GEODATEN\ETRS_1989_UTM_Zone_33N" + \
    r"_7stellen.prj")
ETRS_1989_UTM_Zone_32N = arcpy.SpatialReference("ETRS 1989 UTM Zone 32N")
WGS_84_UTM_ZONE_32N = arcpy.SpatialReference(r"\\mt2.fs.bafg.de\Fachdaten\Projekte\E" + \
    r"lbe_U\EL_000_560_GEK_U2M3\DATENBASIS\GEODATEN\WGS_84_UTM_ZONE_32N.prj")

##
# set the workspace to the enterprise gdb
arcpy.env.workspace = gdb_path

##
# set other arcpy.env properties
arcpy.env.overwriteOutput = True

##
# Create the mosaic datasets DGM001, DGM002_W, DGM010, DGM250
for a_dgm in ["Elbe_DGM001_2007", "Elbe_DGM001_2011", "Elbe_DGM001"]:
    if arcpy.Exists(a_dgm) is not True:
        print "Create the mosaic dataset: " + a_dgm
        arcpy.CreateMosaicDataset_management(gdb_path, a_dgm, ETRS_1989_UTM_Zone_33N, \
                                             1, "64_BIT")

##########
# PROCESSING
##
# fill the prepared mosaic datasets with data

##########
# DGM001
print "DGM001_2007"
# tiles
tiles = []
```

```
##
# Los 1 (ELBE_KM000-290)
print "Los 1 (ELBE_KM000-290)"
folder = r"ELBE_KM000-290\etrs89_grs80_utm33-7+dhhn92\2_dgmw\01m\ascii-grid"

# unzip all packed files to ...
for file in os.listdir(source_path_dgm001 + os.sep + folder):
    if file.endswith(".zip"):
        print "   " + file

        # append path to tiles
        tiles.append(dest_path + os.sep + r"Daten\DGM001\KM_000-290" + os.sep + \
                     file.replace(r".zip", r".grd"))

        # exception for unknown compression method
        # extraction done manually
        if file == r'km056_utm.zip':
            continue

        # skip previously extracted *.grd-files
        if os.path.isfile(dest_path + os.sep + r"Daten\DGM001\KM_000-290" + os.sep + \
                          file.replace(r".zip", r".grd")):
            continue

        # unzip a tile
        #gd_comcom.unzip(source_path_dgm001 + os.sep + folder + os.sep + file, \
        #                dest_path + os.sep + r"Daten\DGM001\KM_000-290")

##
# Los 2 (ELBE_KM290-502)
print ""
print "Los 2 (ELBE_KM290-502)"
folder = r"ELBE_KM290-502\etrs89_grs80_utm33-7+dhhn92\2_dgmw\01m\xyz"
subfolders = [r"km290-300", r"km300-320", r"km320-340", r"km340-360", r"km360-380", \
              r"km380-400", r"km400-420", r"km420-440", r"km440-460", r"km460-480", \
              r"km480-502"]

for a_subfolder in subfolders:

    # rename the subfolder
    b_subfolder = a_subfolder.replace(r"km", r"km_")

    # create the necessary subfolders
    if os.path.isdir(dest_path + os.sep + r"Daten\DGM001\KM_290-502" + os.sep + \
                     b_subfolder) is not True:
        print "Create the subfolder " + b_subfolder
        os.mkdir(dest_path + os.sep + r"Daten\DGM001\KM_290-502" + os.sep + b_subfolder)

    for file in os.listdir(source_path_dgm001 + os.sep + folder + os.sep + a_subfolder):
        if file.endswith(".zip"):
            print "   " + file.replace(".zip", "")

            # file variables
            file_in = dest_path + os.sep + r"Daten\DGM001\KM_290-502" + os.sep + \
                      b_subfolder + os.sep + file.replace(".zip", "")
            file_out = file_in.replace(".xyz", ".asc")

            # continue if files have been produced already
```

```
            if os.path.isfile(file_out):
                # append file_out to tiles
                tiles.append(file_out)
                continue

            # unzip a tile
            #gd_comcom.unzip(source_path_dgm001 + os.sep + folder + os.sep + \
            #                a_subfolder + os.sep + file, dest_path + os.sep + \
            #                r"Daten\DGM001\KM_290-502" + os.sep + b_subfolder)

            # convert *.xyz to *.asc
            xyz2asc.xyz2asc(file_in, -9999, dest_path + os.sep + \
                            r"Daten\DGM001\KM_290-502" + os.sep + b_subfolder)

            # append path to tiles
            tiles.append(file_out)

            # delete intermediate files
            if os.path.isfile(file_in):
                os.remove(file_in)

##
# Los 3 (ELBE_KM502-586)
print ""
print "Los 3 (ELBE_KM502-586)"
folder = r"ELBE_KM502-586\etrs89_grs80_utm33-7+dhhn92\2_dgmw\01m\xyz"
subfolders = [r"km502-520", r"km520-540", r"km540-560", r"km560-580", r"km580-586"]

for a_subfolder in subfolders:

    # rename the subfolder
    b_subfolder = a_subfolder.replace(r"km", "km_")

    # create the necessary subfolders
    if os.path.isdir(dest_path + os.sep + r"Daten\DGM001\KM_502-586" + os.sep + \
                     b_subfolder) is not True:
        print "Create the subfolder " + b_subfolder
        os.mkdir(dest_path + os.sep + r"Daten\DGM001\KM_502-586" + os.sep + b_subfolder)

    for file in os.listdir(source_path_dgm001 + os.sep + folder + os.sep + a_subfolder):
        if file.endswith(".xyz"):
            print "    " + file

            # file variables
            file_in = dest_path + os.sep + r"Daten\DGM001\KM_502-586" + os.sep + \
                      b_subfolder + os.sep + file
            file_out = file_in.replace(".xyz", ".asc")

            # continue if files have been produced already
            if os.path.isfile(file_out):
                # append file_out to tiles
                tiles.append(file_out)
                continue

            # copy a tile
            shutil.copy(source_path_dgm001 + os.sep + folder + os.sep + a_subfolder + \
                        os.sep + file, dest_path + os.sep + r"Daten\DGM001\KM_502-586" + \
                        os.sep + b_subfolder)
```

```
            # convert *.xyz to *.asc
            xyz2asc.xyz2asc(file_in, -9999, dest_path + os.sep + r"Daten\DGM001\KM_50" + \
                            "2-586" + os.sep + b_subfolder)

            # append path to tiles
            tiles.append(file_out)

            # delete intermediate files
            if os.path.isfile(file_in):
                os.remove(file_in)

# import tiles to mosaic-dataset
print "The number of tiles is " + str(len(tiles))
tiles.sort()

# fill the mosaic
arcpy.AddRastersToMosaicDataset_management("Elbe_DGM001_2007", "Raster Dataset", tiles, \
                                           True, True, False, -1, 1, 1, \
                                           ETRS_1989_UTM_Zone_33N_7stellen, "#", False, \
                                           "EXCLUDE_DUPLICATES", True, True, True, \
                                           "ADD RASTER")
# calculate statistics for the mosaic
try:
    arcpy.CalculateStatistics_management("Elbe_DGM001_2007")
except arcpy.ExecuteError as e:
    print e.message()

# set mosaic properties
arcpy.SetMosaicDatasetProperties_management("Elbe_DGM001_2007", \
    rows_maximum_imagesize = 50000, columns_maximum_imagesize = 50000, \
    max_num_per_mosaic = 2500, cell_size = 1, max_num_of_download_items = 2500, \
    data_source_type = "ELEVATION")

##########
# DGM001
print "DGM001_2011"
# tiles
# reuse the list of tiles generated for the DGM001_2007
#tiles = []
# manually remove replaced tiles
remove = [r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_480-502\4847" + \
          r"32625887_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_480-502\485R" + \
          r"32635887_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_480-502\4838" + \
          r"32625886_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_480-502\4831" + \
          r"32635886_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_480-502\483R" + \
          r"32645886_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_480-502\484L" + \
          r"32625885_1m.asc",
```

```
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_480-502\4822" + \
          r"32635885_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_480-502\4818" + \
          r"32645885_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_480-502\483L" + \
          r"32635884_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\4807" + \
          r"32645884_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\4806" + \
          r"32655884_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\479R" + \
          r"32675884_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\478R" + \
          r"32685884_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\4807" + \
          r"32645883_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\4795" + \
          r"32655883_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\4785" + \
          r"32665883_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\4773" + \
          r"32675883_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\4760" + \
          r"32685883_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\480L" + \
          r"32655882_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\4785" + \
          r"32665882_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\4783" + \
          r"32675882_1m.asc",
          r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENBASIS" + \
          r"\GEODATEN\TOPOGRAFIE\HOEHENMODELL\Daten\DGM001\KM_290-502\km_460-480\479L" + \
          r"32685882_1m.asc"]

for an_element in remove:
    if an_element in tiles:
        tiles.remove(an_element)
    else:
        print "The element " + an_element + " was not found in tiles."

# temporarily change workspace
arcpy.env.workspace = source_path_dgm001_2011
```

```
# LOOP
for a_raster in arcpy.ListRasters("*"):
    print "   " + a_raster
    tiles.append(source_path_dgm001_2011 + os.sep + a_raster)

# set the workspace back to the enterprise gdb
arcpy.env.workspace = gdb_path

# import tiles to mosaic-dataset
print "The number of tiles is " + str(len(tiles))
tiles.sort()

# fill the mosaic
arcpy.AddRastersToMosaicDataset_management("Elbe_DGM001_2011", "Raster Dataset", tiles, \
    True, True, False, -1, 1, 1, ETRS_1989_UTM_Zone_33N_7stellen, "#", False, \
    "EXCLUDE_DUPLICATES", True, True, True, "ADD RASTER")
# calculate statistics for the mosaic
try:
    arcpy.CalculateStatistics_management("Elbe_DGM001_2011")
except arcpy.ExecuteError as e:
    print e.message()

# set mosaic properties
arcpy.SetMosaicDatasetProperties_management("Elbe_DGM001_2011", \
    rows_maximum_imagesize = 50000, columns_maximum_imagesize = 50000, \
    max_num_per_mosaic = 2500, cell_size = 1, max_num_of_download_items = 2500, \
    data_source_type = "ELEVATION")

##########
# DGM001
print "DGM001"

arcpy.AddRastersToMosaicDataset_management("Elbe_DGM001", "Raster Dataset", \
    ["Elbe_DGM001_2007", "Elbe_DGM001_2011"], True, True, False, -1, 1, 1, \
    ETRS_1989_UTM_Zone_33N, "#", False, "EXCLUDE_DUPLICATES", True, True, True, \
    "ADD RASTER")

# add year attribute column
arcpy.AddField_management("Elbe_DGM001", "YEAR", "SHORT", "", "", "", "YEAR", "", \
                          "REQUIRED", "")

with arcpy.da.UpdateCursor("Elbe_DGM001", ["NAME", "YEAR"]) as cursor:
    for row in cursor:
        row[1] = int(row[0][-4:])
        cursor.updateRow(row)

# set mosaic properties
arcpy.SetMosaicDatasetProperties_management("Elbe_DGM001", \
    rows_maximum_imagesize = 50000, columns_maximum_imagesize = 50000, \
    max_num_per_mosaic = 2500, cell_size = 1, max_num_of_download_items = 2500, \
    data_source_type = "ELEVATION")

##########
# Quit python
sys.exit(0)
```

4.1.2 River Rhine

The digital elevation model for River Rhine covers the active floodplain of the middle reaches of River Rhine between the Swiss-German border near Basel and the Dutch-German border near Kleve with 1 m spatial resolution. The original datasets were generated in four parts through aerial laser scanning (ALS) for terrestrial parts of the floodplain and echo sounding for aquatic parts of the central water course by the local waterway and navigation autohorities (WSV) between 2003 and 2010 (FUGRO-HGN GmbH, 2011, smile consult GmbH & Inphoris GmbH (2011), Brockmann, Großkordt, & Schumann (2008b), ARGE Vermessung Schmid - Inphoris (2012)). Parts not covered by any of the two data collection methods were filled through linear interpolation. The resulting full mosaic dataset describes the topographic state of 2011.

```
#!/opt/i4/python-2.7.17/bin/python
# -*- coding: utf-8 -*-
##########
import os
import sys
import shutil

try:
    import arcpy
except ImportError as error:
    print "ArcGIS mit dem Modul {0} muss installiert sein.".format(error.message[16:])
    print "Zur vollstaendigen Installation von ArcGIS finden Sie hier eine Beschreibung:"
    print "http://voss-wiki.bafg.de/instanzen/giswiki/doku.php?id=arcgis_106"

###########
# load the i4 module
try:
    sys.path.append(r"\\mt2.fs.bafg.de\Software\Allgemein\GIS\Inform_4_fuer_ArcGIS_10" + \
                    r".x-Entwicklerversion\inform")
    import i4common
    import xyz2asc
except ImportError as error:
    print r"Das Modul {0} kann nicht geladen werden.".format(error.message[16:])
    print r"Stellen Sie bitte sicher, dass das Netzlaufwerk '\\mt2.fs.bafg.de\Fachdat" + \
          r"en' (Z:) von Ihrem PC erreichbar ist."
    sys.exit(1)

###########
# Define variables
# source_paths
source_path_dgm001 = r"\\mt2.fs.bafg.de\Fachdaten\Produkte\Befliegungen_M53\_Rhein"

# dest_path
dest_path = r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Rhein_U\RH_336_860_EKR_U2\DATENBASI" + \
            r"S\GEODATEN\TOPOGRAFIE\HOEHENMODELL"

# gdb_path for the connection data to the Enterprise GDB
gdb_path = i4common.eGDB(readonly=False)

###########
# PREPARATION
##
# Define the coordinate reference systems
ETRS_1989_UTM_Zone_32N = arcpy.SpatialReference("ETRS 1989 UTM Zone 32N")
```

```
ETRS_1989_UTM_Zone_32N_prj = r'PROJCS["ETRS_1989_UTM_Zone_32N",GEOGCS["GCS_ETRS_1989"' + \
                             r',DATUM["D_ETRS_1989",SPHEROID["GRS_1980",6378137.0,298' + \
                             r'.257222101]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0' + \
                             r'17453292519943]],PROJECTION["Transverse_Mercator"],PA' + \
                             r'RAMETER["False_Easting",500000.0],PARAMETER["False_Nor' + \
                             r'thing",0.0],PARAMETER["Central_Meridian",9.0],PARAMETE' + \
                             r'R["Scale_Factor",0.9996],PARAMETER["Latitude_Of_Origin' + \
                             r'",0.0],UNIT["Meter",1.0]]'

##
# set the workspace to this *.gdb
arcpy.env.workspace = gdb_path

##
# set other arcpy.env properties
arcpy.env.overwriteOutput = True

##
# Create the mosaic datasets
for a_dgm in ["Rhein_DGM001_W"]:
    if arcpy.Exists(a_dgm) is not True:
        print "Create the mosaic dataset: " + a_dgm
        arcpy.CreateMosaicDataset_management(gdb_path, a_dgm, ETRS_1989_UTM_Zone_32N, \
                                             1, "64_BIT")

##########
# PROCESSING
##
# fill the prepared mosaic datasets with data

##########
# DGM001
print "DGM001"

## Abschnitt 1 (Rhein 170 - 332)
# Zusatz "oberhalb Iffezheim bis Weil a.R." Kilometer 170 bis 332. Befliegungsdaten aus
# dem Jahr 2003/2004 mit Ergaenzungen neuerer Befliegungsdaten (2004-2010) im
# Gewaesserbett.
#################
# 6-stellige UTM-Koordinaten
#################
print ""
print "Abschnitt 1 (Rhein 170 - 332)"
folder = r"2004-2010_DGM-W_Oberrhein-2_km164-336\1_ETRS89_GRS80_UTM32_DHHN92\1-01_DGM" + \
         r"WA001\ascii-grid"
subfolders = [r"km160-170", r"km170-180", r"km180-190", r"km190-200", r"km200-210", \
              r"km210-220", r"km220-230", r"km230-240", r"km240-250", r"km250-260", \
              r"km260-270", r"km270-280", r"km280-290", r"km290-300", r"km300-310", \
              r"km310-320", r"km320-330", r"km330-340"]
# tiles
tiles = []

for a_subfolder in subfolders:

    # print
    print "  " + a_subfolder

    for file in os.listdir(source_path_dgm001 + os.sep + folder + os.sep + a_subfolder):
```

```
        if file.endswith(".asc"):

            # print
            print "    " + file.replace(".asc", "")

            # append path to tiles
            tiles.append(source_path_dgm001 + os.sep + folder + os.sep + a_subfolder + \
                         os.sep + file)

##
# Abschnitt 1 (Rhein 332 - 560)
# Etwas aeltere Befliegungsdaten, aber Erweiterungen und Modifikationen (2003-2010)
# durch neuere Befliegungsdaten (insbesondere im Gewaesserbett) sind in den Daten
# vorhanden. Format ASCI-Grid. 10-KM-Abschnitte in den Ordnern
#################
# 6-stellige UTM-Koordinaten
#################
print ""
print "Abschnitt 2 (Rhein 332 - 560)"
folder = r"2003-2010_DGM-W_Oberrhein-1_km332-560\1_ETRS89_GRS80_UTM32_DHHN92\1-01_DGM" + \
         r"WA001\ascii-grid"
subfolders = [r"km330-340", r"km340-350", r"km350-360", r"km360-370", r"km370-380", \
              r"km380-390", r"km390-400", r"km400-410", r"km410-420", r"km420-430", \
              r"km430-440", r"km440-450", r"km450-460", r"km460-470", r"km470-480", \
              r"km480-490", r"km490-500", r"km500-510", r"km510-520", r"km520-530", \
              r"km530-540", r"km540-550", r"km550-560"]

for a_subfolder in subfolders:

    # print
    print "  " + a_subfolder

    for file in os.listdir(source_path_dgm001 + os.sep + folder + os.sep + a_subfolder):
        if file.endswith(".asc"):

            # print
            print "    " + file.replace(".asc", "")

            # append path to tiles
            tiles.append(source_path_dgm001 + os.sep + folder + os.sep + a_subfolder + \
            os.sep + file)

##
# Abschnitt 3 (Rhein 560 - 639)
# aeltere Befliegungsdaten aus 2003/04. XYZ-Daten. 20-KM-Abschnitte in den Ordnern
#################
# 7-stellige UTM-Koordinaten in *.xyz, 6-stellig in den ASCII-Grids
#################
print ""
print "Abschnitt 3 (Rhein 560 - 639)"
folder = r"2003-2004_DGM-W_Rhein_km332-643\ETRS89_UTM32\dgmw\A1_1m\xyz"
subfolders = [r"km560-580", r"km580-600", r"km600-620", r"km620-642"]

# create the necessary data folders
if os.path.isdir(dest_path + os.sep + r"Daten\DGM001\KM_560_642") is not True:
    os.mkdir(dest_path + os.sep + r"Daten\DGM001\KM_560_642")

for a_subfolder in subfolders:
```

```
    # print
    print "  " + a_subfolder

    # rename the subfolder
    b_subfolder = a_subfolder.replace(r"km", "km_")

    # create the necessary subfolders
    if os.path.isdir(dest_path + os.sep + r"Daten\DGM001\KM_560_642" + os.sep + \
                     b_subfolder) is not True:
        print "  Create the subfolder " + b_subfolder
        os.mkdir(dest_path + os.sep + r"Daten\DGM001\KM_560_642" + os.sep + b_subfolder)

    for file in os.listdir(source_path_dgm001 + os.sep + folder + os.sep + a_subfolder):
        if file.endswith(".xyz"):

            # file variables
            file_in = source_path_dgm001 + os.sep + folder + os.sep + a_subfolder + \
                      os.sep + file
            file_out = dest_path + os.sep + r"Daten\DGM001\KM_560_642" + os.sep + \
                       b_subfolder + os.sep + file.replace(".xyz", ".asc")
            prj_out = dest_path + os.sep + r"Daten\DGM001\KM_560_642" + os.sep + \
                      b_subfolder + os.sep + file.replace(".xyz", ".prj")

            # continue if files have been produced already
            if os.path.isfile(file_out):
                # print
                print "    " + file

                # append file_out to tiles
                tiles.append(file_out)
                continue

            # print
            print "    Convert " + file

            # convert *.xyz to *.asc
            xyz2asc.xyz2asc(file_in, 2000, -9999, dest_path + os.sep + \
                            r"Daten\DGM001\KM_560_642" + os.sep + b_subfolder)

            # create *.prj-file
            with open(prj_out, "w") as prj_file:
                prj_file.write(ETRS_1989_UTM_Zone_32N_prj)

            # append path to tiles
            tiles.append(file_out)

##
# Abschnitt 4 (Rhein 639 - 892)
# Alle neuesten Informationen, insbesondere im Gewaesserbett, sind in den Daten
# vorhanden. Befliegungen 2009/10. Format ASCI-Grid. 1-Kilometer-Abschnitte
#################
# 8-stellige UTM-Koordinaten, manually reduced to 6-digits
#################
print ""
print "Abschnitt 4 (Rhein 639 - 892)"
folder = r"2009-2010_DGM-W_Niederrhein_km639-892\ETRS89_GRS80_UTM32_DHHN92\05_DGM-W\5" + \
         r".04.1_Modell_A_1m_ascii_1x1"
```

```
# create the necessary folder for 6 digit ascii-tiles
if os.path.isdir(dest_path + os.sep + r"Daten\DGM001\KM_639-892") is not True:
    print "Create the subfolder " + b_subfolder
    os.mkdir(dest_path + os.sep + r"Daten\DGM001\KM_639-892")

for file in os.listdir(source_path_dgm001 + os.sep + folder):
    if file.endswith(".asc"):

        # print
        print "    " + file

        # file variables
        file_in = source_path_dgm001 + os.sep + folder + os.sep + file
        file_out = dest_path + os.sep + r"Daten\DGM001\KM_639-892" + os.sep + file
        prj_out = dest_path + os.sep + r"Daten\DGM001\KM_639-892" + os.sep + \
                  file.replace(".asc", ".prj")

        # continue if files have been produced already
        if os.path.isfile(file_out):
            # print
            print "    " + file

            # append file_out to tiles
            tiles.append(file_out)
            continue

        i = 0
        with open(file_out, "w") as new_file:
            with open(file_in, "r") as old_file:
                for line in old_file:
                    if i == 2:
                        xllcorner = str(int(line.replace("xllcorner ", "")) - 32000000)
                        new_file.write("xllcorner " + xllcorner + "\n")
                    else:
                        new_file.write(line)
                    i = i + 1

        # create *.prj-file
        with open(prj_out, "w") as prj_file:
            prj_file.write(ETRS_1989_UTM_Zone_32N_prj)

        # append path to tiles
        tiles.append(file_out)

# import tiles to mosaic-dataset
print "The number of tiles is " + str(len(tiles))
print ""
tiles.sort()

# fill the mosaic
arcpy.AddRastersToMosaicDataset_management("Rhein_DGM001_W", "Raster Dataset", tiles, \
    True, True, False, -1, 1, 1, ETRS_1989_UTM_Zone_32N, "#", False, \
    "EXCLUDE_DUPLICATES", True, True, True, "ADD RASTER")

# calculate statistics for the mosaic
try:
    arcpy.CalculateStatistics_management("Rhein_DGM001_W")
except arcpy.ExecuteError as e:
```

```
    print e.message()

# set mosaic properties
arcpy.SetMosaicDatasetProperties_management("Rhein_DGM001_W", \
    rows_maximum_imagesize = 50000, columns_maximum_imagesize = 50000, \
    max_num_per_mosaic = 2500, cell_size = 1, max_num_of_download_items = 2500, \
    data_source_type = "ELEVATION")

##########
# Quit python
sys.exit(0)
```

4.2 Tiling

To reduce the number of individual datasets for later processing the mosaic datasets had to be split into a manageable number of datasets with a manageable size.

```
#!/opt/i4/python-2.7.17/bin/python
# -*- coding: utf-8 -*-
##########
# load python modules
import arcpy
import math
import sys
import os
import dirsync

##########
# load the i4 module
try:
    sys.path.append(r"\\mt2.fs.bafg.de\Software\Allgemein\GIS\Inform_4_fuer_ArcGIS_10" + \
                    r".x-Entwicklerversion\inform")
    import i4common
except ImportError as error:
    print r"Das Modul {0} kann nicht geladen werden.".format(error.message[16:])
    print r"Stellen Sie bitte sicher, dass das Netzlaufwerk '\\mt2.fs.bafg.de\Fachdat" + \
          r"en' (Z:) von Ihrem PC erreichbar ist."
    sys.exit(1)

##########
# check ArcGIS license level and set to ArcInfo
if arcpy.ProductInfo() != "ArcInfo":
    if arcpy.CheckProduct("ArcInfo") == u"Available":
        if arcpy.SetProduct("ArcInfo") == u"CheckedOut":
            print "\nEs wurde eine ArcInfo-Lizenz bezogen.\n"
        else:
            print "\nEs ist keine ArcInfo-Lizenz verfuegbar, weshalb das Skript nicht" + \
                  " ausgefuehrt werden kann.\n"
            sys.exit(1)
    elif arcpy.CheckProduct("ArcInfo") == u"AlreadyInitialized":
        print "\nEine ArcInfo-Lizenz ist bereits in Benutzung.\n"
    else:
        print "\nEs ist keine ArcInfo-Lizenz verfuegbar, weshalb das Skript nicht aus" + \
              "gefuehrt werden kann.\n"
        sys.exit(1)
else:
```

```
    print "\nEs wurde eine ArcInfo-Lizenz bezogen.\n"

##########
# Enterprise GDB
e_gdb = i4common.eGDB(readonly=False)
if os.path.exists(e_gdb) is not True:
    print "Zur Ausfuehrung dieses Skriptes benoetigen Sie ArcGIS for Desktop 10.6"
    print "und eine Verbindungsdatei zur Enterprise-Geodatabase der BfG. Da Ihnen"
    print "diese fehlt, bricht dieses Skript nun ab."
    sys.exit(1)

################################################################################
################################################################################
# river
if 'river' not in locals():
    river = r"Elbe"
################################################################################
################################################################################

##########
# set other variables
if river == "Elbe":
    ##
    # gdb_path
    gdb_path = r"EL_000_586_UFD"

    ##
    # coordinate reference system
    crs = arcpy.SpatialReference("ETRS 1989 UTM Zone 33N")

    ##
    # upper end
    cross_section_traces_upper_end_km = 0
    cross_section_traces_upper_end = 0
    ##
    # lower end
    cross_section_traces_lower_end_km = 586
    cross_section_traces_lower_end = 586
    gauging_station_lower_name = "GEESTHACHT_WEHR"

elif river == "Rhein":
    ##
    # gdb_path
    gdb_path = r"RH_336_867_UFD"

    ##
    # coordinate reference system
    crs = arcpy.SpatialReference("ETRS 1989 UTM Zone 32N")

    ##
    # upper end
    cross_section_traces_upper_end_km = 336.21
    cross_section_traces_upper_end = 336.21
    ##
    # lower end
    cross_section_traces_lower_end_km = 865.7
    cross_section_traces_lower_end = 865.7
    gauging_station_lower_name = "EMMERICH"
```

```
else:
    print "ERROR: river"
    sys.exit(1)

# set river letter and gdb
river_letter = river[0].lower()
network_gdb = r"\\mt2.fs.bafg.de\Fachdaten\U\U3\Auengruppe_INFORM" + os.sep + gdb_path + \
              os.sep + "data" + os.sep + gdb_path + ".gdb"
gdb = r"E:" + os.sep + gdb_path + os.sep + "data" + os.sep + gdb_path + ".gdb"

print "Fluss: " + river
print ""

##########
# create file geodatabase
if arcpy.Exists(gdb) is not True:
    arcpy.CreateFileGDB_management(os.path.split(gdb)[0], os.path.split(gdb)[1], \
                                   "CURRENT")
    print 'Die File-Geodatabase "' + gdb + '" wurde erstellt.\n'

##########
# env settings
arcpy.env.workspace = gdb
arcpy.env.overwriteOutput = True
arcpy.env.outputCoordinateSystem = crs

##########
# create a temporary copy of "BFG.*_Aktive_Aue" and "BFG.*_Pegel"
if arcpy.Exists("active_floodplain") is not True:
    arcpy.CopyFeatures_management(e_gdb + os.sep + "BFG." + river + "_Aktive_Aue", \
                                  "active_floodplain")
if arcpy.Exists("gauging_station_data") is not True:
    arcpy.CopyFeatures_management(e_gdb + os.sep + "BFG." + river + "_Pegel", \
                                  "gauging_station_data")

##########
# make active feature layers
arcpy.MakeFeatureLayer_management(e_gdb + os.sep + r"BFG." + river + "_Querprofilspuren", \
                                  "cross_section_traces")
arcpy.MakeFeatureLayer_management(r"gauging_station_data", "gauging_station_data")

##########
# km_values of gauging_station_data
km_values = [row[0] for row in arcpy.da.SearchCursor("gauging_station_data", "km_qps", \
    "tiles = \'true\'", sql_clause=[None, "order by km_qps"])]
ids = [int(row[0]) for row in arcpy.da.SearchCursor("gauging_station_data", \
    ["id", "km_qps"], "tiles = \'true\'", sql_clause=[None, "order by km_qps"])]
gauging_stations = [row[0] for row in arcpy.da.SearchCursor("gauging_station_data", \
    ["gauging_station_flut3", "km_qps"], "tiles = \'true\'", \
    sql_clause=[None, "order by km_qps"])]
del row

##########
# create empty polygon feature class for the resulting tiles
arcpy.CreateFeatureclass_management(gdb, "tiles", "POLYGON", "", "DISABLED", "DISABLED", \
                                    crs)

# add the necessary columns to the 'temp' feature class
```

```
arcpy.AddField_management("tiles", "id", "LONG", "", "", "", "id", "", "", "")
arcpy.AddField_management("tiles", "name", "TEXT", "", "", "30", "name", "", "", "")
arcpy.AddField_management("tiles", "name_km", "TEXT", "", "", "30", "name_km", "", "", "")
arcpy.AddField_management("tiles", "from_km", "DOUBLE", "", "", "", "from_km", "", "", "")
arcpy.AddField_management("tiles", "to_km", "DOUBLE", "", "", "", "to_km", "", "", "")
arcpy.AddField_management("tiles", "gauging_station_upper", "TEXT", "", "", "50", \
                          "gauging_station_upper", "", "", "")
arcpy.AddField_management("tiles", "gauging_station_lower", "TEXT", "", "", "50", \
                          "gauging_station_lower", "", "", "")
arcpy.AddField_management("tiles", "ram_size_flut3", "DOUBLE", 10, 7, "", \
                          "ram_size_flut3", "", "", "")
arcpy.AddField_management("tiles", "ram_size_cat", "LONG", "", "", "", "ram_size_cat", \
                          "", "", "")

# create empty polygon feature class for the resulting tiles
arcpy.CreateFeatureclass_management(gdb, "active_floodplain_sections", "POLYGON", "", \
                                    "DISABLED", "DISABLED", crs)

# add the necessary columns to the 'temp' feature class
arcpy.AddField_management("active_floodplain_sections", "id", "LONG", "", "", "", "id", \
                          "", "", "")
arcpy.AddField_management("active_floodplain_sections", "name", "TEXT", "", "", "30", \
                          "name", "", "", "")
arcpy.AddField_management("active_floodplain_sections", "name_km", "TEXT", "", "", \
                          "30", "name_km", "", "", "")
arcpy.AddField_management("active_floodplain_sections", "from_km", "DOUBLE", "", "", "", \
                          "from_km", "", "", "")
arcpy.AddField_management("active_floodplain_sections", "to_km", "DOUBLE", "", "", "", \
                          "to_km", "", "", "")
arcpy.AddField_management("active_floodplain_sections", "gauging_station_upper", "TEXT",\
                           "", "", "50", "gauging_station_upper", "", "", "")
arcpy.AddField_management("active_floodplain_sections", "gauging_station_lower", "TEXT", \
                          "", "", "50", "gauging_station_lower", "", "", "")
arcpy.AddField_management("active_floodplain_sections", "ram_size_flut3", "DOUBLE", 10, \
                          7, "", "ram_size_flut3", "", "", "")
arcpy.AddField_management("active_floodplain_sections", "ram_size_cat", "LONG", "", "", \
                          "", "ram_size_cat", "", "", "")

##########
# loop over all sections
print "Prozessierung der Kacheln:"
for a_section in range(len(gauging_stations)):

    # set missing variables for the inserts
    id = a_section + 1
    id_string = river_letter + str(id).zfill(3)

    # print
    print "%5s" % str(id) + ": " + gauging_stations[a_section]

    # IFFEZHEIM AND SCHOENA1
    if a_section == 0:
        ###
        # select fixed cross_section_traces_upper
        cross_section_traces_upper_km = cross_section_traces_upper_end_km
        cross_section_traces_upper = cross_section_traces_upper_end

        ###
```

```
        # select lower cross_section_traces
        cross_section_traces_lower_km = int(math.ceil(km_values[a_section + 1]))
        cross_section_traces_lower_sections = int(km_values[a_section + 1] * 1000)
        cross_section_traces_lower = int(km_values[a_section + 1] * 1000 + 500)

        ### for the sections
        # select cross_section_traces based on the "STATION_INT"-column and write into
        # "cross_section_traces_lower"
        arcpy.SelectLayerByAttribute_management("cross_section_traces", "NEW_SELECTION", \
            ' "STATION_INT" = ' + str(cross_section_traces_lower_sections))
        arcpy.CopyFeatures_management("cross_section_traces", \
        "tmp_cross_section_traces_lower")
        arcpy.SelectLayerByAttribute_management ("cross_section_traces", \
            "CLEAR_SELECTION")

        # buffer cross_section_traces_upper
        arcpy.Buffer_analysis("tmp_cross_section_traces_lower", \
            "tmp_cross_section_traces_lower_buffered", "0.005 Meters")

        # erase a strip from the 'active_floodplain'
        arcpy.Erase_analysis("active_floodplain", \
            "tmp_cross_section_traces_lower_buffered", \
            "tmp_active_floodplain_multipart")

        ### for the tiles
        # select cross_section_traces based on the "STATION_INT"-column and write into
        # "cross_section_traces_lower"
        with arcpy.da.SearchCursor("cross_section_traces", ["STATION_INT"], \
            where_clause=' "STATION_INT" >= ' + str(cross_section_traces_lower) + \
            ' AND "STATION_INT" < ' + str(cross_section_traces_lower + 500), \
            sql_clause=(None, 'ORDER BY STATION_INT DESC')) as cursor:
            for row in cursor:
                cross_section_traces_lower = row[0]
        arcpy.SelectLayerByAttribute_management("cross_section_traces", "NEW_SELECTION", \
            ' "STATION_INT" = ' + str(cross_section_traces_lower))
        arcpy.CopyFeatures_management("cross_section_traces",
            "temp_cross_section_traces_lower")
        arcpy.SelectLayerByAttribute_management ("cross_section_traces", \
            "CLEAR_SELECTION")

        # buffer cross_section_traces_upper
        arcpy.Buffer_analysis("temp_cross_section_traces_lower", \
            "temp_cross_section_traces_lower_buffered", "0.005 Meters")

        # erase a strip from the 'active_floodplain'
        arcpy.Erase_analysis("active_floodplain", \
            "temp_cross_section_traces_lower_buffered", \
            "temp_active_floodplain_multipart")

    # ARTLENBURG AND REES3
    elif a_section == len(gauging_stations) - 1:
        ###
        # select cross_section_traces_upper
        cross_section_traces_upper_km = int(math.floor(km_values[a_section]))
        cross_section_traces_upper_sections = int(km_values[a_section] * 1000)
        cross_section_traces_upper = int(km_values[a_section] * 1000 - 500)

        if cross_section_trace_lower_sections <> cross_section_traces_upper_sections:
```

```
            cross_section_traces_upper_sections = cross_section_trace_lower_sections

        ### for the sections
        # select cross_section_traces based on the "STATION_INT"-column and write into
        # "cross_section_traces_upper"
        arcpy.SelectLayerByAttribute_management("cross_section_traces", "NEW_SELECTION", \
            ' "STATION_INT" = ' + str(cross_section_traces_upper_sections))
        arcpy.CopyFeatures_management("cross_section_traces", \
            "tmp_cross_section_traces_upper")
        arcpy.SelectLayerByAttribute_management ("cross_section_traces", \
            "CLEAR_SELECTION")

        # buffer cross_section_traces_upper
        arcpy.Buffer_analysis("tmp_cross_section_traces_upper", \
            "tmp_cross_section_traces_upper_buffered", "0.005 Meters")

        # erase the upper stripe from the 'active_floodplain'
        arcpy.Erase_analysis("active_floodplain", \
            "tmp_cross_section_traces_upper_buffered", \
            "tmp_active_floodplain_multipart")

        ### for the tiles
        # select cross_section_traces based on the "STATION_INT"-column and write into
        # "cross_section_traces_upper"
        with arcpy.da.SearchCursor("cross_section_traces", ["STATION_INT"], \
            where_clause=' "STATION_INT" <= ' + str(cross_section_traces_upper) + \
            ' AND "STATION_INT" > ' + str(cross_section_traces_upper - 500), \
            sql_clause=(None, 'ORDER BY STATION_INT ASC')) as cursor:
            for row in cursor:
                cross_section_traces_upper = row[0]
        arcpy.SelectLayerByAttribute_management("cross_section_traces", "NEW_SELECTION", \
            ' "STATION_INT" = ' + str(cross_section_traces_upper))
        arcpy.CopyFeatures_management("cross_section_traces", \
            "temp_cross_section_traces_upper")
        arcpy.SelectLayerByAttribute_management ("cross_section_traces", \
            "CLEAR_SELECTION")

        # buffer cross_section_traces_upper
        arcpy.Buffer_analysis("temp_cross_section_traces_upper", \
            "temp_cross_section_traces_upper_buffered", "0.005 Meters")

        # erase the upper stripe from the 'active_floodplain'
        arcpy.Erase_analysis("active_floodplain", \
            "temp_cross_section_traces_upper_buffered", \
            "temp_active_floodplain_multipart")

        ###
        # select fixed cross_section_traces_lower
        cross_section_traces_lower_km = cross_section_traces_lower_end_km
        cross_section_traces_lower = cross_section_traces_lower_end

    else:
        ###
        # select cross_section_traces_upper
        cross_section_traces_upper_km = int(math.floor(km_values[a_section]))
        cross_section_traces_upper_sections = int(km_values[a_section] * 1000)
        cross_section_traces_upper = int(km_values[a_section] * 1000 - 500)
```

```
        if cross_section_trace_lower_sections <> cross_section_traces_upper_sections:
            cross_section_traces_upper_sections = cross_section_trace_lower_sections

        ### for the sections
        # select cross_section_traces based on the "STATION_INT"-column and write into
        # "cross_section_traces_upper"
        arcpy.SelectLayerByAttribute_management("cross_section_traces", "NEW_SELECTION", \
            ' "STATION_INT" = ' + str(cross_section_traces_upper_sections))
        arcpy.CopyFeatures_management("cross_section_traces", \
            "tmp_cross_section_traces_upper")
        arcpy.SelectLayerByAttribute_management ("cross_section_traces", \
            "CLEAR_SELECTION")

        # buffer cross_section_traces_upper
        arcpy.Buffer_analysis("tmp_cross_section_traces_upper", \
            "tmp_cross_section_traces_upper_buffered", "0.005 Meters")

        # erase the upper stripe from the 'active_floodplain'
        arcpy.Erase_analysis("active_floodplain", \
            "tmp_cross_section_traces_upper_buffered", "tmp_active_floodplain_upper")

        ### for the tiles
        # select cross_section_traces based on the "STATION_INT"-column and write into
        # "cross_section_traces_upper"
        with arcpy.da.SearchCursor("cross_section_traces", ["STATION_INT"], \
            where_clause=' "STATION_INT" <= ' + str(cross_section_traces_upper) + \
            ' AND "STATION_INT" > ' + str(cross_section_traces_upper - 500), \
            sql_clause=(None, 'ORDER BY STATION_INT ASC')) as cursor:
            for row in cursor:
                cross_section_traces_upper = row[0]
        arcpy.SelectLayerByAttribute_management("cross_section_traces", "NEW_SELECTION", \
            ' "STATION_INT" = ' + str(cross_section_traces_upper))
        arcpy.CopyFeatures_management("cross_section_traces", \
            "temp_cross_section_traces_upper")
        arcpy.SelectLayerByAttribute_management ("cross_section_traces", \
            "CLEAR_SELECTION")

        # buffer cross_section_traces_upper
        arcpy.Buffer_analysis("temp_cross_section_traces_upper", \
            "temp_cross_section_traces_upper_buffered", "0.005 Meters")

        # erase the upper stripe from the 'active_floodplain'
        arcpy.Erase_analysis("active_floodplain", \
            "temp_cross_section_traces_upper_buffered", "temp_active_floodplain_upper")

        ###
        # select cross_section_traces_lower
        cross_section_traces_lower_km = int(math.ceil(km_values[a_section + 1]))
        cross_section_traces_lower_sections = int(km_values[a_section + 1] * 1000)
        cross_section_traces_lower = int(km_values[a_section + 1] * 1000 + 500)

        ### for the sections
        # select cross_section_traces based on the "STATION_INT"-column and write into
        # "cross_section_traces_lower"
        arcpy.SelectLayerByAttribute_management("cross_section_traces", "NEW_SELECTION", \
            ' "STATION_INT" = ' + str(cross_section_traces_lower_sections))
        arcpy.CopyFeatures_management("cross_section_traces", \
            "tmp_cross_section_traces_lower")
```

```
        arcpy.SelectLayerByAttribute_management ("cross_section_traces", \
            "CLEAR_SELECTION")

        # buffer cross_section_traces_upper
        arcpy.Buffer_analysis("tmp_cross_section_traces_lower", \
            "tmp_cross_section_traces_lower_buffered", "0.005 Meters")

        # erase the lower stripe from the 'active_floodplain'
        arcpy.Erase_analysis("tmp_active_floodplain_upper", \
            "tmp_cross_section_traces_lower_buffered", \
            "tmp_active_floodplain_multipart")

        ### for the tiles
        # select cross_section_traces based on the "STATION_INT"-column and write into
        # "cross_section_traces_lower"
        with arcpy.da.SearchCursor("cross_section_traces", ["STATION_INT"], \
            where_clause=' "STATION_INT" >= ' + str(cross_section_traces_lower) + \
            ' AND "STATION_INT" < ' + str(cross_section_traces_lower + 500), \
            sql_clause=(None, 'ORDER BY STATION_INT DESC')) as cursor:
            for row in cursor:
                cross_section_traces_lower = row[0]
        arcpy.SelectLayerByAttribute_management("cross_section_traces", "NEW_SELECTION", \
            ' "STATION_INT" = ' + str(cross_section_traces_lower))
        arcpy.CopyFeatures_management("cross_section_traces", \
            "temp_cross_section_traces_lower")
        arcpy.SelectLayerByAttribute_management ("cross_section_traces", \
            "CLEAR_SELECTION")

        # buffer cross_section_traces_upper
        arcpy.Buffer_analysis("temp_cross_section_traces_lower", \
            "temp_cross_section_traces_lower_buffered", "0.005 Meters")

        # erase the lower stripe from the 'active_floodplain'
        arcpy.Erase_analysis("temp_active_floodplain_upper", \
            "temp_cross_section_traces_lower_buffered", \
            "temp_active_floodplain_multipart")

    # make the sections multipart polygon a feature layer
    arcpy.MultipartToSinglepart_management("tmp_active_floodplain_multipart", \
            "tmp_active_floodplain")
    arcpy.MakeFeatureLayer_management(r"tmp_active_floodplain", \
            "tmp_active_floodplain")

    # split the tiles multipart polygon into single part polygons and make it a feature
    # layer
    arcpy.MultipartToSinglepart_management("temp_active_floodplain_multipart", \
            "temp_active_floodplain")
    arcpy.MakeFeatureLayer_management(r"temp_active_floodplain", \
            "temp_active_floodplain")

    # select the upper gauging_station and the polygon containing this upper
    # gauging_station
    ## SELECT Polygon by the location of the upper gauging_station
    if a_section == 0:
        arcpy.SelectLayerByAttribute_management("gauging_station_data", "NEW_SELECTION", \
            ' "gauging_station_flut3" = \'' + gauging_stations[a_section] + '\' ')
        arcpy.SelectLayerByLocation_management("tmp_active_floodplain", "CONTAINS", \
            "gauging_station_data", "", "NEW_SELECTION")
```

```
        arcpy.SelectLayerByLocation_management("temp_active_floodplain", "CONTAINS", \
            "gauging_station_data", "", "NEW_SELECTION")

    elif a_section == (len(gauging_stations) - 1):
        arcpy.SelectLayerByAttribute_management("gauging_station_data", "NEW_SELECTION", \
            ' "gauging_station_flut3" = \'' + gauging_stations[a_section] + '\' ')
        arcpy.SelectLayerByLocation_management("tmp_active_floodplain", "CONTAINS", \
            "gauging_station_data", "", "NEW_SELECTION")
        arcpy.SelectLayerByLocation_management("tmp_active_floodplain", "", "", "", \
            "SWITCH_SELECTION")
        arcpy.SelectLayerByLocation_management("temp_active_floodplain", "CONTAINS", \
            "gauging_station_data", "", "NEW_SELECTION")

    elif a_section == (len(gauging_stations) - 2) and river == "Rhein":
        arcpy.SelectLayerByAttribute_management("gauging_station_data", "NEW_SELECTION", \
            ' "id" < ' + str(ids[a_section]) + ' ')
        arcpy.SelectLayerByAttribute_management("gauging_station_data", \
            "ADD_TO_SELECTION", ' "id" > ' + str(ids[a_section + 1]) + ' ')
        arcpy.SelectLayerByLocation_management("tmp_active_floodplain", "CONTAINS", \
            "gauging_station_data", "", "NEW_SELECTION")
        arcpy.SelectLayerByLocation_management("tmp_active_floodplain", "", "", "", \
            "SWITCH_SELECTION")

        arcpy.SelectLayerByAttribute_management ("gauging_station_data", \
            "CLEAR_SELECTION")
        arcpy.SelectLayerByAttribute_management("gauging_station_data", "NEW_SELECTION", \
            ' "gauging_station_flut3" = \'' + gauging_stations[a_section] + '\' ')
        arcpy.SelectLayerByLocation_management("temp_active_floodplain", "CONTAINS", \
            "gauging_station_data", "", "NEW_SELECTION")

    else:
        # select all gauging stations except gauging_stations[a_section] and
        # gauging_stations[a_section + 1]
        # select all parts of the temp_active_floodplain that do not contain the
        # previsously selected gauging_stations
        arcpy.SelectLayerByAttribute_management("gauging_station_data", "NEW_SELECTION", \
            ' "id" < ' + str(ids[a_section]) + ' ')
        arcpy.SelectLayerByAttribute_management("gauging_station_data", \
            "ADD_TO_SELECTION", ' "id" > ' + str(ids[a_section + 1]) + ' ')
        arcpy.SelectLayerByLocation_management("tmp_active_floodplain", "CONTAINS", \
            "gauging_station_data", "", "NEW_SELECTION")
        arcpy.SelectLayerByLocation_management("tmp_active_floodplain", "", "", "", \
            "SWITCH_SELECTION")
        arcpy.SelectLayerByLocation_management("temp_active_floodplain", "CONTAINS", \
            "gauging_station_data", "", "NEW_SELECTION")
        arcpy.SelectLayerByLocation_management("temp_active_floodplain", "", "", "", \
            "SWITCH_SELECTION")

    # clear gauging_station_data
    arcpy.SelectLayerByAttribute_management ("gauging_station_data", "CLEAR_SELECTION")

    # gauging_station_name
    gauging_station_name = gauging_stations[a_section].replace("-", "_").replace(" ", \
                                                                                  "_")

    # Save active_floodplain_section_*
    arcpy.CopyFeatures_management("tmp_active_floodplain", id_string + "_" + \
        gauging_station_name + "_active_floodplain_section")
```

```
    arcpy.Dissolve_management(id_string + "_" + gauging_station_name + \
        "_active_floodplain_section", "tmp_active_floodplain_section_" + \
        gauging_station_name, "ORIG_FID")
    arcpy.Buffer_analysis("tmp_active_floodplain_section_" + gauging_station_name, \
        "tmp_active_floodplain_section_" + gauging_station_name + "_buffered", \
        "0.005 Meters")

    arcpy.Delete_management(id_string + "_" + gauging_station_name + \
        "_active_floodplain_section")
    arcpy.Rename_management("tmp_active_floodplain_section_" + gauging_station_name + \
        "_buffered", id_string + "_" + gauging_station_name + \
        "_active_floodplain_section")

    # Save active_floodplain_*
    arcpy.CopyFeatures_management("temp_active_floodplain", id_string + "_" + \
        gauging_station_name + "_active_floodplain")

    # set missing variables for the inserts
    name = river_letter + unicode(str(id).zfill(3)) + u"_" + \
           unicode(gauging_station_name)
    if a_section == (len(gauging_stations) - 1):
        name_km = str(int(km_values[a_section])).zfill(4) + "_" + \
                  str(round(cross_section_traces_lower_end_km, 0)).zfill(4)
    else:
        name_km = str(int(km_values[a_section])).zfill(4) + "_" + \
                  str(int(km_values[a_section + 1])).zfill(4)
    from_km = km_values[a_section]
    gauging_station_upper = gauging_stations[a_section]

    if a_section == (len(gauging_stations) - 1):
        to_km = cross_section_traces_lower_end_km
        gauging_station_lower = gauging_station_lower_name
    else:
        to_km = km_values[a_section + 1]
        gauging_station_lower = gauging_stations[a_section + 1]

    extent = arcpy.Describe(id_string + "_" + gauging_station_name + \
                            "_active_floodplain_section").extent
    ram_size_flut3 = round((math.ceil(extent.XMax) - math.floor(extent.XMin)) * \
                           (math.ceil(extent.YMax) - math.floor(extent.YMin)) * \
                            22 / 1000000000, 3)
    if ram_size_flut3 >= 10:
        ram_size_cat = 3
    elif ram_size_flut3 < 10 and ram_size_flut3 >= 5.0:
        ram_size_cat = 2
    elif ram_size_flut3 < 5.0 and ram_size_flut3 >= 2.0:
        ram_size_cat = 1
    else:
        ram_size_cat = 0

    ###
    # manipulate the attribute table of *_active_floodplain_section
    arcpy.DeleteField_management(id_string + "_" + gauging_station_name + \
        "_active_floodplain_section", "ORIG_FID")
    arcpy.DeleteField_management(id_string + "_" + gauging_station_name + \
        "_active_floodplain_section", "BUFF_DIST")
    arcpy.AddField_management(id_string + "_" + gauging_station_name + \
        "_active_floodplain_section", "id", "LONG", "", "", "", "id", "", "", "")
```

```
arcpy.AddField_management(id_string + "_" + gauging_station_name + \
    "_active_floodplain_section", "name", "TEXT", "", "", "30", "name", "", "", "")
arcpy.AddField_management(id_string + "_" + gauging_station_name + \
    "_active_floodplain_section", "name_km", "TEXT", "", "", "30", "name_km", "", \
    "", "")
arcpy.AddField_management(id_string + "_" + gauging_station_name + \
    "_active_floodplain_section", "from_km", "DOUBLE", "", "", "", "from_km", "", \
    "", "")
arcpy.AddField_management(id_string + "_" + gauging_station_name + \
    "_active_floodplain_section", "to_km", "DOUBLE", "", "", "", "to_km", "", "", \
    "")
arcpy.AddField_management(id_string + "_" + gauging_station_name + \
    "_active_floodplain_section", "gauging_station_upper", "TEXT", "", "", "50", \
    "gauging_station_upper", "", "", "")
arcpy.AddField_management(id_string + "_" + gauging_station_name + \
    "_active_floodplain_section", "gauging_station_lower", "TEXT", "", "", "50", \
    "gauging_station_lower", "", "", "")
arcpy.AddField_management(id_string + "_" + gauging_station_name + \
    "_active_floodplain_section", "ram_size_flut3", "DOUBLE", 10, 7, "", \
    "ram_size_flut3", "", "", "")
arcpy.AddField_management(id_string + "_" + gauging_station_name + \
    "_active_floodplain_section", "ram_size_cat", "LONG", "", "", "", \
    "ram_size_cat", "", "", "")

# update line in *_active_floodplain_section
with arcpy.da.UpdateCursor(id_string + "_" + gauging_station_name + \
    "_active_floodplain_section", ["id", "name", "name_km", "from_km", "to_km", \
    "gauging_station_upper", "gauging_station_lower", "ram_size_flut3", \
    "ram_size_cat"]) as cursor:
    for row in cursor:
        cursor.updateRow([id, name, name_km, from_km, to_km, gauging_station_upper, \
                          gauging_station_lower, ram_size_flut3, ram_size_cat])

# delete temporary objects
del cursor

# merge active_floodplain_section_* to active_floodplain_sections
arcpy.Merge_management(["active_floodplain_sections", id_string + "_" + \
    gauging_station_name + "_active_floodplain_section"], \
    "temp_active_floodplain_sections")
arcpy.Delete_management("active_floodplain_sections")
arcpy.Rename_management("temp_active_floodplain_sections", \
    "active_floodplain_sections")

###
# Get the extent of the active_floodplain_*
extent = arcpy.Describe(id_string + "_" + gauging_station_name + \
                        "_active_floodplain").extent
ram_size_flut3 = round((math.ceil(extent.XMax) - math.floor(extent.XMin)) * \
                       (math.ceil(extent.YMax) - math.floor(extent.YMin)) * \
                       22 / 1000000000, 3)

# use the extent to construct a rectangular polygon
polygon = arcpy.Polygon(arcpy.Array([arcpy.Point(math.floor(extent.XMin),
                                                 math.ceil(extent.YMax)),
                                     arcpy.Point(math.ceil(extent.XMax),
                                                 math.floor(extent.YMax)),
                                     arcpy.Point(math.ceil(extent.XMax),
```

```
                                                    math.floor(extent.YMin)),
                                          arcpy.Point(math.floor(extent.XMin),
                                                    math.ceil(extent.YMin))]))

    # insert line in tiles
    with arcpy.da.InsertCursor("tiles", ["SHAPE@", "id", "name", "name_km", "from_km", \
        "to_km", "gauging_station_upper", "gauging_station_lower", "ram_size_flut3", \
        "ram_size_cat"]) as cursor:
        cursor.insertRow([polygon, id, name, name_km, from_km, to_km, \
                          gauging_station_upper, gauging_station_lower, \
                          ram_size_flut3, ram_size_cat])

    # delete temporary objects
    del extent, cursor

    for a_featureclass in arcpy.ListFeatureClasses("temp_*"):
        arcpy.Delete_management(a_featureclass)
    for a_featureclass in arcpy.ListFeatureClasses("tmp_*"):
        arcpy.Delete_management(a_featureclass)

    # remember cross_section_traces_lower in separate variable for the next loop
    cross_section_trace_lower_sections = cross_section_traces_lower_sections

##########
# save important results in the enterprise GDB
arcpy.CopyFeatures_management("active_floodplain_sections", e_gdb + os.sep + "BFG." + \
                              river + "_Aktive_Aue_Abschnitte")
arcpy.CopyFeatures_management("tiles", e_gdb + os.sep + "BFG." + river + "_Kacheln")

##########
# exit Python
sys.exit(0)
```

This automatic splitting of the active floodplain and its elevational data resulted in 49 tiles for River Elbe and 40 tiles for River Rhein:

4.3 Export as Arc/Info ASCII

```
#!/opt/i4/python-2.7.17/bin/python
# -*- coding: utf-8 -*-
##########
# load python modules
import arcpy
import sys
import os
import math
from datetime import datetime as dt

##########
# load the i4 module
try:
    sys.path.append(r"\\mt2.fs.bafg.de\Software\Allgemein\GIS\Inform_4_fuer_ArcGIS_10" + \
                    r".x-Entwicklerversion\inform")
    import i4common
except ImportError as error:
```

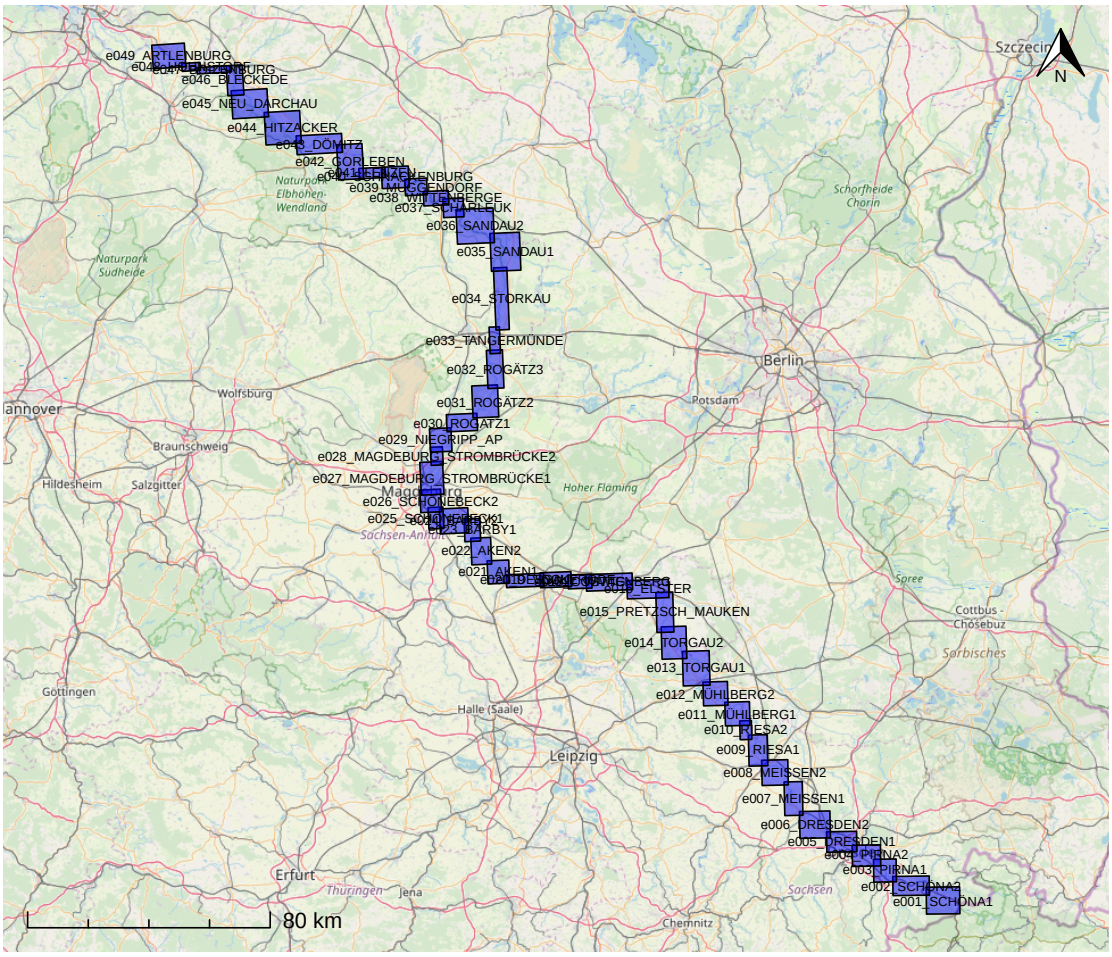

Figure 1: Tiling for the digital elevation models along River Elbe. Background map © OpenStreetMap contributors (Data available from https://www.openstreetmap.org).

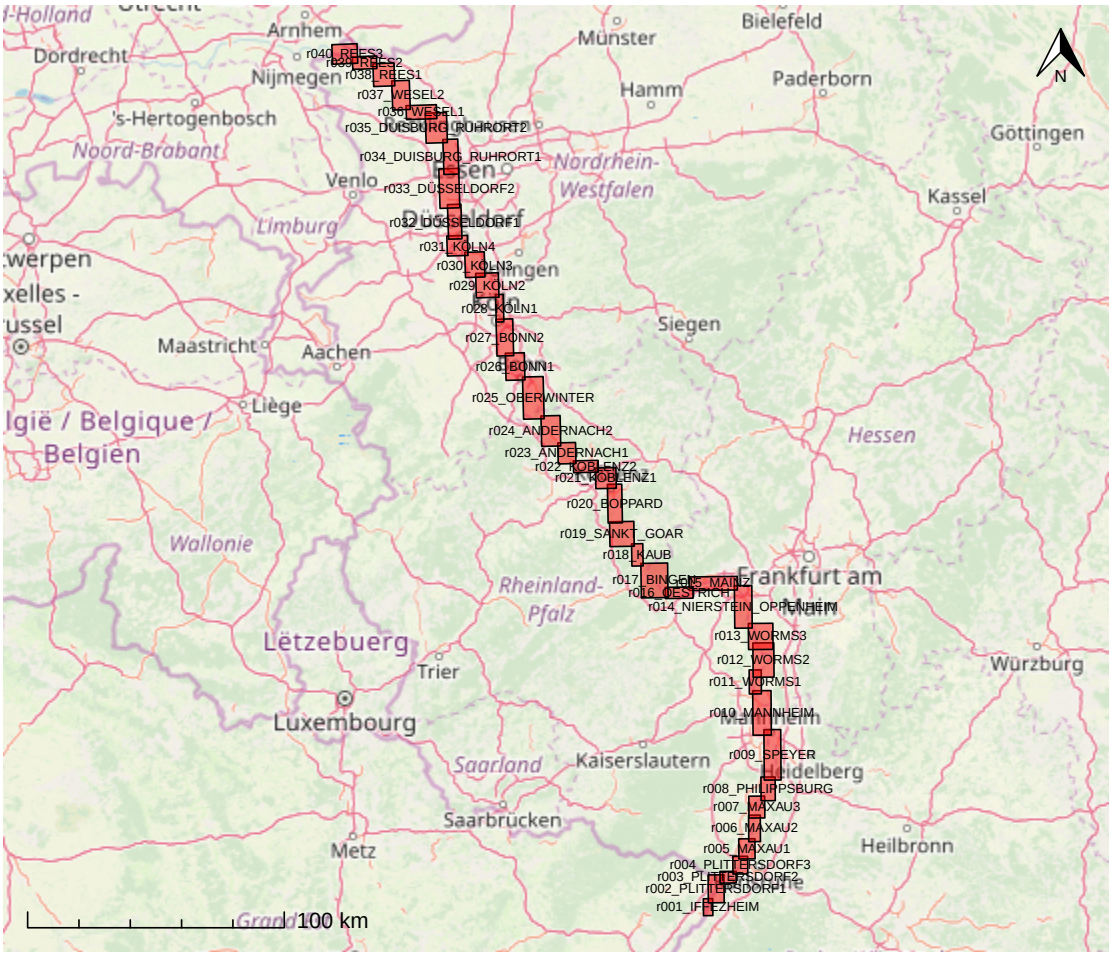

Figure 2: Tiling for the digital elevation models along River Rhein. Background map © OpenStreetMap contributors (Data available from https://www.openstreetmap.org).

```
    print r"Das Modul {0} kann nicht geladen werden.".format(error.message[16:])
    print r"Stellen Sie bitte sicher, dass das Netzlaufwerk '\\mt2.fs.bafg.de\Fachdat" + \
          r"en' (Z:) von Ihrem PC erreichbar ist."
    sys.exit(1)

##########
# Enterprise GDB
e_gdb = i4common.eGDB(readonly=False)
if os.path.exists(e_gdb) is not True:
    print "Zur Ausfuehrung dieses Skriptes benoetigen Sie ArcGIS for Desktop 10.6"
    print "und eine Verbindungsdatei zur Enterprise-Geodatabase der BfG. Da Ihnen"
    print "diese fehlt, bricht dieses Skript nun ab."
    sys.exit(1)

################################################################################
################################################################################
# river
if 'river' not in locals():
    river = r"Elbe"
################################################################################
################################################################################

##########
# set other variables
if river == r"Elbe":
    ##
    # gdb_path
    gdb_path = r"EL_000_586_UFD"

    ##
    # coordinate reference system
    crs = arcpy.SpatialReference("ETRS 1989 UTM Zone 33N")

    ##
    # dem
    dem = e_gdb + os.sep + r"BFG." + river + r"_DEM001"
    if arcpy.Exists(dem) is not True:
        dem = r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Elbe_U\EL_000_560_GEK_U2M3\DATENB" + \
              r"ASIS\GEODATEN\TOPOGRAFIE\HOEHENMODELL\DGM.gdb\DGM001"
elif river == r"Rhein":
    ##
    # gdb_path
    gdb_path = r"RH_336_867_UFD"

    ##
    # coordinate reference system
    crs = arcpy.SpatialReference("ETRS 1989 UTM Zone 32N")

    ##
    # dem
    dem = e_gdb + os.sep + r"BFG." + river + r"_DGM001_W"
    if arcpy.Exists(dem) is not True:
        dem = r"\\mt2.fs.bafg.de\Fachdaten\Projekte\Rhein_U\RH_336_860_EKR_U2\DATENB" + \
              r"ASIS\GEODATEN\TOPOGRAFIE\HOEHENMODELL\DGM.gdb\DGM001"
else:
    print "ERROR: river"
    sys.exit(1)
```

```
# set river letter, gdb and out
#network_gdb = r"\\mt2.fs.bafg.de\Fachdaten\U\U3\Auengruppe_INFORM" + os.sep + \
#gdb_path + os.sep + "data" + os.sep + gdb_path + ".gdb"
gdb = r"E:" + os.sep + gdb_path + os.sep + "data" + os.sep + gdb_path + ".gdb"

out = r"\\mt2.fs.bafg.de\Fachdaten\U\U3\Auengruppe_INFORM" + os.sep + gdb_path + \
    os.sep + r"data\ascii"

print ""
print "Fluss: " + river
print ""

##########
# env settings
arcpy.env.workspace = gdb
arcpy.env.overwriteOutput = True
arcpy.env.outputCoordinateSystem = crs
arcpy.env.snapRaster = dem

if arcpy.Exists("temp"):
    arcpy.Delete_management("temp")
##########
# loop over all sections
if arcpy.Exists(e_gdb + os.sep + "BFG." + river + "_Aktive_Aue_Abschnitte"):
    sections = [row[0] for row in arcpy.da.SearchCursor(e_gdb + os.sep + "BFG." + river + \
                    "_Aktive_Aue_Abschnitte", field_names=["name", "id"], \
                    sql_clause = (None, "ORDER BY id ASC") )]
else:
    sections = [row[0] for row in arcpy.da.SearchCursor(gdb + os.sep + \
                    "active_floodplain_sections", field_names=["name", "id"], \
                    sql_clause = (None, "ORDER BY id ASC") )]
del row

print "Prozessierung der Kacheln"

for a_section in sections:

    print "   " + a_section

    area_desc = arcpy.Describe(a_section + "_active_floodplain_section")
    area_extent = str(math.floor(area_desc.extent.XMin)) + " " + \
                  str(math.floor(area_desc.extent.YMin)) + " " + \
                  str(math.ceil(area_desc.extent.XMax)) + " " + \
                  str(math.ceil(area_desc.extent.YMax))

    arcpy.env.Extent = area_extent
    print "           extent: " + area_extent

    # export DEM
    if os.path.isfile(out + os.sep + a_section + "_DEM.asc") is not True:
        # export DEM
        print "     DEM"
        arcpy.Clip_management(e_gdb + os.sep + "BFG." + river + "_DGM001", area_extent, \
                              "temp", "", "", "", "NO_MAINTAIN_EXTENT")
        dem_desc = arcpy.Describe("temp")
        dem_extent = str(dem_desc.extent.XMin) + " " + str(dem_desc.extent.YMin) + " " + \
                     str(dem_desc.extent.XMax) + " " + str(dem_desc.extent.YMax)
        print "       dem extent: " + dem_extent
```

```
        print "       dem   ncol: " + str(dem_desc.width)
        print "       dem   nrow: " + str(dem_desc.height)
        print "       dem   resx: " + str(dem_desc.meanCellWidth)
        print "       dem   resy: " + str(dem_desc.meanCellHeight)
        arcpy.RasterToASCII_conversion("temp", out + os.sep + a_section + "_DEM.asc")
        os.remove(out + os.sep + a_section + "_DEM.asc.xml")
        os.remove(out + os.sep + a_section + "_DEM.prj")

        # capitalize file name
        os.rename(out + os.sep + a_section + "_DEM.asc", \
                  out + os.sep + a_section + "_DEM.asc.new")
        b_section = a_section[:5] + a_section[5:].upper()
        os.rename(out + os.sep + a_section + "_DEM.asc.new", \
                  out + os.sep + b_section + "_DEM.asc")

##########
# clean up
arcpy.Delete_management("temp")

##########
# replace decimal separator , with .
print ""
print "Prozessierung der DEMs"

ascii_files = os.listdir(out)

# exclude DEM- and CSA-datasets
for a_ascii_file in ascii_files:
    ts = os.path.getmtime(out + os.sep + a_ascii_file)).strftime("%Y-%m-%d")
    if a_ascii_file.endswith("DEM.asc") and \
        dt.fromtimestamp(ts == dt.now().strftime("%Y-%m-%d"):
        print "  " + a_ascii_file
        fname = os.path.join(out + os.sep + a_ascii_file)
        inFile = open(fname, "r")
        outFile = open(fname + ".new", "w")
        for line in inFile:
            newline = line.replace(",",".")
            outFile.write(newline)
        inFile.close()
        outFile.close()
        os.remove(fname)
        os.rename(fname + ".new", fname)

##########
# exit Python
sys.exit(0)
```

4.4 Conversion to GeoTIFF

```
#!/opt/i4/python-2.7.17/bin/python
import os
import sys
#import subprocess

################################################################################
################################################################################
```

```
# river
if 'river' not in locals():
    river = r"Elbe"
################################################################################
################################################################################

##########
# set other variables
if river == r"Elbe":
    ##
    # gdb_path
    gdb_path = r"EL_000_586_UFD"

elif river == r"Rhein":
    ##
    # gdb_path
    gdb_path = r"RH_336_867_UFD"

else:
    print "ERROR: river"
    sys.exit(1)

in_path = r"\\mt2.fs.bafg.de\Fachdaten\U\U3\Auengruppe_INFORM" + os.sep + gdb_path + \
          os.sep + r"data\ascii"
ou_path = r"\\mt2.fs.bafg.de\Fachdaten\U\U3\Auengruppe_INFORM" + os.sep + gdb_path + \
          os.sep + r"data\tiff"

##########
# conversion
for root, dirs, files in os.walk(in_path):
    for a_file in files:
        if a_file.endswith("_DEM.asc"):
            in_file = os.path.join(root, a_file)
            ou_file = os.path.join(ou_path, a_file.replace(".asc", ".tif"))
            if os.path.exists(ou_file) is not True:
                print(in_file)
                if river == "Elbe":
                    crs = "EPSG:25833"
                else:
                    crs = "EPSG:25832"
                os.system("gdal_translate -q -ot Float64 -co \"COMPRESS=LZW\" -of \"" + \
                          "GTiff\" -a_srs " + crs + " " + in_file + " " + ou_file)
                #p = subprocess.Popen(["gdal_translate", in_file, ou_file, "-ot", \
                #                      "Float64", "-co", "\"COMPRESS=LZW\"", "-of", \
                #                      "\"GTiff\"", "-a_srs", crs, "-q"], shell = True)
                #p.wait()

##########
# exit Python
sys.exit(0)
```

5 Products

The python scripts included in this documentation produce a number of datasets building up on each other. Most of the datasets are actually BfG-internal datasets which are not publicly visible, but at least the final processing step provides GeoTiff products, which have been published on pangaea.de.

To summarize the outcomes, we provide some additional numbers highlighting some of the key information:

5.1 River Elbe

- 1033 individual raster datasets were merged into one mosaic dataset
- a tiling dataset with 49 tiles was generated
- the mosaic dataset was exported into 49 Arc/Info ASCII Grid datasets
- these 49 Arc/Info ASCII Grid datasets were converted and compressed to GeoTIFF datasets
- these 49 GeoTIFF datasets were uploaded to pangaea.de and are available through the following DOI: 10.1594/PANGAEA.919293

Table 1: Exported GeoTiff's along River Elbe.

tile name	river km	area (km^2)	file size (MB)
e001_SCHOENA1	0 - 16	104	69.9
e002_SCHOENA2	16 - 35	80	78.4
e003_PIRNA1	35 - 45	59	92.9
e004_PIRNA2	45 - 56	69	95.1
e005_DRESDEN1	56 - 70	70	120.0
e006_DRESDEN2	70 - 82	93	102.3
e007_MEISSEN1	82 - 96	70	80.4
e008_MEISSEN2	96 - 108	75	52.9
e009_RIESA1	108 - 122	66	81.5
e010_RIESA2	122 - 128	25	56.6
e011_MUEHLBERG1	128 - 142	66	84.6
e012_MUEHLBERG2	142 - 154	65	89.3
e013_TORGAU1	154 - 172	104	88.8

tile name	river km	area (km^2)	file size (MB)
e014_TORGAU2	172 - 186	89	71.5
e015_PRETZSCH_MAUKEN	186 - 200	75	128.2
e016_ELSTER	200 - 214	84	119.9
e017_WITTENBERG	214 - 236	85	163.4
e018_COSWIG	236 - 246	36	81.2
e019_VOCKERODE	246 - 261	51	124.1
e020_DESSAU	261 - 275	54	119.9
e021_AKEN1	275 - 285	55	65.3
e022_AKEN2	285 - 295	58	88.6
e023_BARBY1	295 - 301	39	52.2
e024_BARBY2	301 - 312	78	76.8
e025_SCHOENEBECK1	312 - 319	35	37.3
e026_SCHOENEBECK2	319 - 327	51	50.1
e027_MAGDEBURG_STROMBRUECKE1	327 - 338	82	92.4
e028_MAGDEBURG_STROMBRUECKE2	338 - 344	24	34.7
e029_NIEGRIPP_AP	344 - 351	55	51.5
e030_ROGAETZ1	351 - 361	57	55.3
e031_ROGAETZ2	361 - 375	89	66.5
e032_ROGAETZ3	375 - 388	66	77.8
e033_TANGERMUENDE	388 - 396	30	33.7
e034_STORKAU	396 - 416	89	90.2
e035_SANDAU1	416 - 429	119	74.3
e036_SANDAU2	429 - 447	136	95.3
e037_SCHARLEUK	447 - 454	40	49.3
e038_WITTENBERGE	454 - 464	38	38.1
e039_MUEGGENDORF	464 - 475	38	45.3
e040_SCHNACKENBURG	475 - 485	61	62.9
e041_LENZEN	485 - 492	29	35.3
e042_GORLEBEN	492 - 505	93	85.4
e043_DOEMITZ	505 - 523	90	99.1
e044_HITZACKER	523 - 536	120	68.0
e045_NEU_DARCHAU	536 - 550	105	86.2
e046_BLECKEDE	550 - 558	50	59.5
e047_BOIZENBURG	558 - 569	22	35.2
e048_HOHNSTORF	569 - 574	15	12.1
e049_ARTLENBURG	574 - 586	80	54.1

5.2 River Rhine

- 5900 individual raster datasets were merged into one mosaic dataset
- a tiling dataset with 40 tiles was generated
- the mosaic dataset was exported into 40 Arc/Info ASCII Grid datasets
- these 40 Arc/Info ASCII Grid datasets were converted and compressed to GeoTIFF datasets
- these 40 GeoTIFF datasets were uploaded to pangaea.de and are available through the following DOI: 10.1594/PANGAEA.919308

Table 2: Exported GeoTiff's along River Rhein.

tile name	river km	area (km^2)	file size (MB)
r001_IFFEZHEIM	336 - 340	26	46.6
r002_PLITTERSDORF1	340 - 350	72	57.8
r003_PLITTERSDORF2	350 - 356	30	29.0
r004_PLITTERSDORF3	356 - 362	42	43.9
r005_MAXAU1	362 - 371	53	55.1
r006_MAXAU2	371 - 380	50	57.5
r007_MAXAU3	380 - 389	56	93.8
r008_PHILIPPSBURG	389 - 401	54	79.2
r009_SPEYER	401 - 425	134	164.4
r010_MANNHEIM	425 - 443	128	152.2
r011_WORMS1	443 - 453	45	63.2
r012_WORMS2	453 - 470	112	120.4
r013_WORMS3	470 - 481	100	137.6
r014_NIERSTEIN_OPPENHEIM	481 - 498	115	123.5
r015_MAINZ	498 - 518	101	131.4
r016_OESTRICH	518 - 528	48	75.5
r017_BINGEN	528 - 546	143	88.9
r018_KAUB	546 - 556	39	40.7
r019_SANKT_GOAR	556 - 570	93	55.3
r020_BOPPARD	570 - 592	85	72.4
r021_KOBLENZ1	592 - 603	65	60.7
r022_KOBLENZ2	603 - 614	45	63.7
r023_ANDERNACH1	614 - 623	57	38.2
r024_ANDERNACH2	623 - 637	88	66.7
r025_OBERWINTER	637 - 655	131	109.8
r026_BONN1	655 - 667	77	89.5
r027_BONN2	667 - 688	92	151.6
r028_KOELN1	688 - 700	31	61.4
r029_KOELN2	700 - 713	82	151.3

tile name	river km	area (km^2)	file size (MB)
r030_KOELN3	713 - 730	73	113.8
r031_KOELN4	730 - 744	61	88.7
r032_DUESSELDORF1	744 - 760	69	84.6
r033_DUESSELDORF2	760 - 781	114	132.3
r034_DUISBURG_RUHRORT1	781 - 797	76	84.0
r035_DUISBURG_RUHRORT2	797 - 814	93	113.2
r036_WESEL1	814 - 824	61	82.4
r037_WESEL2	824 - 837	74	96.8
r038_REES1	837 - 848	69	98.0
r039_REES2	848 - 856	44	56.9
r040_REES3	856 - 866	70	82.7

6 References

ARGE Vermessung Schmid - Inphoris. (2012). Aufbau eines Digitalen Geländemodells des Niederrheinwasserlaufes (DGM-W Niederrhein). Technical Report, Klosterneuburg, Austria.

Bivand, R., & Rundel, C. (2019). *rgeos: Interface to Geometry Engine - Open Source ('GEOS')*. https://CRAN.R-project.org/package=rgeos

Bivand, R., Keitt, T., & Rowlingson, B. (2019). *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*. https://CRAN.R-project.org/package=rgdal

Brockmann, H., & Schumann, L. (2012). Produktblatt: DGM-W Elbe-Lenzen, 2003-2011. Data Sheet, Bundesanstalt für Gewässerkunde, Koblenz, Germany.

Brockmann, H., Großkordt, U., & Schumann, L. (2008a). Auswertung digitaler Fernerkundungsaufnahmen des Elbe-Wasserlaufes (FE-Datenauswertung Elbe). BfG-Bericht, Bundesanstalt für Gewässerkunde, Koblenz, Germany.

Brockmann, H., Großkordt, U., & Schumann, L. (2008b). Digitales Geländemodell des Rhein-Wasserlaufes von Iffezheim bis Bonn (DGM-W Rhein). BfG-Bericht, Bundesanstalt für Gewässerkunde, Koblenz, Germany.

Chamberlain, S., Woo, K., MacDonald, A., Zimmerman, N., & Simpson, G. (2020). *Pangaear: Client for the 'pangaea' database*. https://CRAN.R-project.org/package=pangaear

Cheng, J., Karambelkar, B., & Xie, Y. (2019). *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*. https://CRAN.R-project.org/package=leaflet

Dunnington, D. (2017). *prettymapr: Scale bar, north arrow, and pretty margins in r*. https://CRAN.R-project.org/package=prettymapr

Dunnington, D. (2019). *rosm: Plot raster map tiles from open street map and other sources*. https://CRAN.R-project.org/package=rosm

ESRI. (2018). *ArcGIS Desktop: The mapping and analytics platform*. https://desktop.arcgis.com

FUGRO-HGN GmbH. (2011). Aufbau eines Digitalen Geländemodells des Oberrheinwasserverlaufes (DGM-W Oberrhein-2, Basel bis Iffezheim). Technical Report, Torgau, Germany.

GDAL/OGR contributors. (2019). *GDAL/OGR geospatial data abstraction software library*. Open Source Geospatial Foundation. https://gdal.org

Python Software Foundation. (2017). *Python Programming Language*. Delaware, US: Python

Software Foundation. https://www.python.org

R Core Team. (2019). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/

Rossum, G. van. (1995). *Python tutorial* (No. CS-R9526). Amsterdam, Netherlands: Centrum voor Wiskunde en Informatica (CWI).

smile consult GmbH & Inphoris GmbH. (2011). DGM-W Oberrhein 1. Technical Report, Hannover, Germany.

Ushey, K., Allaire, J., & Tang, Y. (2019). *reticulate: Interface to 'Python'*. https://CRAN.R-project.org/package=reticulate

Wasserstraßen- und Schifffahrtsverwaltung des Bundes (WSV). (2016). Digitales Geländemodell des Wasserlaufs (DGM-W). https://www.govdata.de/web/guest/suchen/-/details/digitales-gelandemodell-des-wasserlaufs-dgm-w

Weber, A. (2020a). Digital elevation model (DEM 1) of the River Elbe floodplain between Schmilka and Geesthacht, Germany. https://doi.org/https://doi.org/10.1594/PANGAEA.919293

Weber, A. (2020b). Digital elevation model (DEM 1) of the River Rhine floodplain between Iffezheim and Kleve, Germany. https://doi.org/https://doi.org/10.1594/PANGAEA.919308

Wickham, H. (2019). *stringr: Simple, consistent wrappers for common string operations*. https://CRAN.R-project.org/package=stringr

Xie, Y. (2014). Knitr: A comprehensive tool for reproducible research in R. In V. Stodden, F. Leisch, & R. D. Peng (Eds.), *Implementing Reproducible Computational Research*. Chapman; Hall/CRC. http://www.crcpress.com/product/isbn/9781466561595

Xie, Y. (2015). *Dynamic Documents with R and knitr* (2nd ed.). Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.org/knitr/

Xie, Y. (2020). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. https://yihui.org/knitr/