Welcher Typ sind Sie?

Die Skriptsprache erweitert capella um ungeahnte Möglichkeiten, und zwar für jedermann: Wir glauben, dass es im wesentlichen drei Typen von capella-Anwendern gibt, die alle auf ihre Weise aus der Skriptsprache ihren Nutzen ziehen. In einem kleinen Beispiel schildern wir, was Sie davon haben:

Der Programmierexperte...

....schreibt sich selbst Skripte für die Spezialfunktionen, die er in capella vermisst. Nehmen wir also einmal an, dass er ein Skript für die automatische Umwandlung seiner Partitur in eine Gitarrentabulatur schreibt.

Der Computerfreund...

...kann selbst nicht programmieren, ist aber halbwegs in der Lage, ein Skript zu lesen, wenn er dessen Aufgabe kennt. Damit kann er das Skript für seine eigenen Bedürfnisse umfrisieren. Auf diese Weise macht er aus der Gitarrentabulatur eine Banjo- oder Lautentabulatur.

Der Musiker...

... will mit der Welt der Skripte erst einmal nichts zu tun haben. Er lädt sich die neuen capella-Funktionen „Gitarren-, Banjo oder Lautentabulatur" aus dem Internet und fügt sie auf komfortable Weise als Plugin in sein capella ein. Jetzt kann er die Tabulaturen über das Menü benutzen, ohne eine einzige Programmzeile zu kennen.

capella-Skripte selbst programmieren

Dieser Abschnitt ist für neugierige capella-Anwender mit XML- und Programmierkennnissen bestimmt.

Mit capella seit Version 5 wird ein Entwicklerhandbuch (online) ausgeliefert. Darin wird ausführlich auf XML und die CapXML-Spezifikation sowie die Programmierung von Python-Skripts eingegangen. Sie können auch die zahlreichen Skripte selbst studieren - die Beispielskripte im Lieferumfang von capella oder ein Besuch auf der capella-Website bieten reichhaltiges Material!

Hier können Sie einen kurzen Überblick über capella und seine Skriptsprache gewinnen.

Interne Skripts

Diese Skripts können nur von capella aus gestartet werden.

Beispiel 1

Mit den folgenden drei Zeilen wird in jedem System der aktuellen Partitur die zweite Notenzeile um eine Quinte transponiert:

```
for s in activeScore().systems():
    if s.nStaves() >= 2:
        s.staff(1).transpose(5)
```

Hinweis: mit s.staff(1) wird die zweite Notenzeile angesprochen, weil die Zählung (wie in Java oder C++) bei Null beginnt.

Beispiel 2

Das folgende Skript hat sich intern in der Praxis bereits bewährt: Mit einem einzigen Auf-

ruf dieses Skripts wurden (in etwa 20 Minuten) mehrere Tausend Partituren der Digitalen Partiturbibliothek, die noch im alten capella-2-Format vorlagen, ins Format von capella 2000/2002 konvertiert.

```
„""Konvertierung von Dateien im capella-2-Format ins capella-3-Format."""
#
#  targetDir muss existieren!
#  Darin werden die Unterverzeichnisse cap2 und cap3 angelegt.
#  Diese übernehmen die Verzeichnisstruktur von sourceDir.
#
#  capella-3-Dateien werden nach cap3 kopiert,
#  capella-2-Dateien werden nach cap2 kopiert und nach cap3 konvertiert.
#
#  Die konvertierten Dateien übernehmen das Dateidatum von den Originalen.

#--- Die folgenden Zeilen anpassen! ---
sourceDir = r'D:\Temp\DigPB\Partituren'
targetDir = r'D:\Temp\DigPB\Konvert'

#-------------------------------------------------------------------------------
import os, os.path, shutil

def isCap2File(path):
    „""ist die Datei <path> im capella-2-Format?"""
    #  erste 4 Bytes: 0x74B3E289, 0x74B3E28A oder 0x74B3E28B
    s = file(path).read(4)
    return s[1:4] == ,\xE2\xB3\x74' and abs(ord(s[0])-0x8A) <= 1

def convert(source, target3, target2):
    „"" konvertiert bzw. (rekursiv) alle Dateien aus <source>"""
    #   Dateien im cap3-Format werden nach <target3> kopiert,
    #   Dateien im cap2-Format werden nach <target3> konvertiert
    #                              und nach <target2> kopiert
    if os.path.isdir(source):
        # Unterverzeichnis rekursiv bearbeiten
        if not os.path.exists(target3):  os.mkdir(target3)
        if not os.path.exists(target2):  os.mkdir(target2)
        for relName in os.listdir(source):
            convert(os.path.join(source, relName),
                    os.path.join(target3, relName),
                    os.path.join(target2, relName))
    else:
        if isCap2File(source):
            if not os.path.exists(target2):
                shutil.copy2(source, target2)
                cap.openScore(source)
                activeScore().write(target3, 0)
                attr = os.stat(source)
                os.utime(target3, attr[7:9])      # Datum kopieren
                cap.closeActiveScore()
        else:
            if not os.path.exists(target2):
                shutil.copy2(source, target3)
```

Externe Skripts

Diese Skripts werden unabhängig von capella gestartet und greifen direkt auf eine Partiturdatei zu.

Beispiel 3

Dieses Python-Skript zeigt exemplarisch, wie sich CapXML-Dateien mit Standard-Werkzeu-

gen manipulieren lassen.

Der größte Teil dieses Skripts (schwarze Schrift) umfasst das Entpacken der komprimierten CapXML-Datei und den Aufbau des XML-Baums mit Hilfe des Document-Object Modells (DOM).

All dies wird im Entwicklerhandbuch ausführlich erklärt.

```
# alle Bindebögen der Partitur in gestrichelte umwandeln

import xml.dom.minidom, string, zipfile, os

def changeSlur(doc, slur):
    „""Bindebogen gestrichelt machen"""
    form = slur.getElementsByTagName(‚form')
    if len(form) == 0:
       form = doc.createElement(‚form')
       slur.appendChild(form)
    else: form = form[0]
    form.setAttribute(‚endWidth', ‚0.0625') # alle Maße in Notenlinienabständen
    form.setAttribute(‚midWidth', ‚0.0625')
    form.setAttribute(‚dotDist', ‚1')
    form.setAttribute(‚dotWidth', ‚0.75')

def copyElement(doc, el, f):
    „""kopiert das Element (rekursiv) mit Ausnahme der <slur>-Elemente"""
    if el.tagName == ‚slur':
        changeSlur(doc, el)
    f.write(‚<' + el.tagName)
    if el.hasAttributes():
        a = el.attributes
        for i in range(a.length):
            f.write(‚ %s="%s"' % (a.item(i).name, el.getAttribute(a.item(i).
     name)))
    if el.hasChildNodes() or el.tagName == ‚textarea':
        f.write (‚>')                          # Start-Tag schließen
        for n in el.childNodes:
            if n.nodeType == n.TEXT_NODE:
                f.write (n.data)
            elif n.nodeType == n.ELEMENT_NODE:
                copyElement (doc, n, f)
        f.write (‚</%s>' % el.tagName)
    else:
        f.write (‚/>')                         # Element ohne Inhalt

def transform (inputFile, outputFile):
    „""alle Dateien aus Eingabearchiv in Ausgabearchiv kopieren,
       dabei ‚score.xml' bearbeiten"""
    zr = zipfile.ZipFile (inputFile, ‚r')
    zw = zipfile.ZipFile (outputFile, ‚w')
    for name in zr.namelist():
        t = zr.read (name)
        if name == ‚score.xml':
            doc = xml.dom.minidom.parseString (t)
            #--- Partitur ändern und in temporärer Datei speichern
            f = os.tmpfile()
            copyElement (doc, doc.documentElement, f)
            f.seek(0)
            t = f.read()
            f.close()
        info = zipfile.ZipInfo (name)
        info.compress_type = zipfile.ZIP_DEFLATED
        zw.writestr (info, t)
```

```
#-------------------------------------------------------------------------------
     ------
# Hauptprogramm:

transform(r'c:\capella\beispiel.cpx', ,c:\capella\beispiel-x.cpx')
```

Beispiel 4

Natürlich lassen sich externe Skripts auch von internen Skripts aufrufen.

Im folgenden Listing sehen Sie, wie man Beispiel 3 in capella integrieren kann.

Nach Aufnahme in das Plugin-Menü (und evtl. in die Symbolleiste) wird daraus für den Anwender ein ganz normaler Befehl, der sogar wieder rückgängig gemacht werden kann!

```
#--- bis hier wie in Beispiel 3 ---
#-------------------------------------------------------------------------------
     ------
# Hauptprogramm:

import tempfile

if activeScore():
    tempInput = tempfile.mktemp(„.cpx")
    tempOutput = tempfile.mktemp(„.cpx")
    activeScore().write(tempInput)       # Partitur in temporäre Datei speichern
    transform(tempInput, tempOutput)     # diese Datei wie oben extern
     bearbeiten...
    activeScore().read(tempOutput)       # ...und zurückholen
    os.remove(tempInput)
    os.remove(tempOutput)

In der endgültigen Version von capella 5 wird vieles noch wesentlich einfacher
     werden. So lässt sich etwa im letzten Beispiel der Umgang mit den externen
     Dateien in eine Hilfsklasse packen, die gleich mitgeliefert wird.
Dann würde das Beispiel etwa so aussehen;

#--- bis hier wie in Beispiel 3 ---
#-------------------------------------------------------------------------------
     ------
# Hauptprogramm:

if activeScore():
    t = activeScore().capTransformer()
    transform(t.input, t.output)
```

Auch das Entpacken und Packen könnte hierin bereits gekapselt sein und damit auch das Beispiel 1 vereinfachen.