Site-Plugin anlegen Grundlagen: (Stand 2014)

1. Eine neue .py Datei im Ordner *addons/plugin.video.xstream/sites* erstellen. z.B. *test.py*

2. Mittels eines Texteditors folgende Zeilen Code in die Datei schreiben:

Code:

```
SITE_IDENTIFIER = 'test' SITE_NAME = 'MeinTest'  def load():     pass
```

SITE_IDENTIFIER muss mit dem Dateinamen übereinstimmen!
SITE_NAME ist lediglich der im Menu angezeigte Name.
Die Funktion *load()* muss immer implementiert werden und beinhaltet üblicher Weise das Hautmenu des jeweiligen Site-Plugins.

3.Fertig im xStream Hauptmenu sollte sich nun ein neuer Punkt "MainTest" befinden.

Wählt man nun "MeinTest" im Hauptmenu aus wird die Funktion *load()* der test.py aufgerufen.
Da sie jedoch nur das Statement *pass* beinhaltet tut sie genau gar nichts.
Wie man die Funktion *load()* und weitere Funktionen mit Code befüllt, welcher tatsächlich etwas tut**,** folgt im nächsten Abschnitt.

Site-Plugin programmieren:

In diesem Abschnitt soll gezeigt werden wie man ein Site-Plugin programmieren kann, welches tatsächlich sinvolle Ausgaben erzeugt. Das ganze wird vorläufig am Beispiel der Seite kinoleak.tv erfolgen.

Es wird benötigt:

- Texteditor, möglichst mit Syntaxhighlighting und Zeilennummern (z.B. Notepad++)
- ein Webbrowser, möglichst mit Entwicklerwerkzeugen oder einem Addon wie z.B. FireBug
- (optional) anstatt des Texteditors ein IDE (z.B. VisualStudio mit Pythonerweiterung, Eclipse mit Pythonerweiterung, o.ä.)

Außerdem is es hilfreich wenn man selbständig Scriptfehler, die im .log auftauchen, erkennen oder googlen kann, falls man sich mal verschrieben hat.

Wir benennen die *test.py* aus dem vorherigen Abschnitt in *kinoleak.py* um und passen SITE_IDENTIFIER und SITE_NAME ebefalls an.

Wenn das Site-Plugin ein Icon ehalten soll können wir außerdem die Variable SITE_ICON anlegen. Das zugehörige Bild muss unter *plugin.video.xstream/resources/art/sites* abgelegt werden.

Die Adresse der Seite weisen wir der Variable URL_MAIN zu, da wir diese öfter in mehreren Funktionen gebrauchen können.

Hauptmenu:

Um mittels der Funktion load() das Hauptmenu dieses Site-Plugin zu erzeugen müssen noch einige module (*cGui* und *cGuiElement*) importiert werden.

Zusättzlich importieren wir noch *cRequestHandler*, *cParser* und *ParameterHandler*. Diese werden wir später noch benötigen.
Das ganze setzen wir, auch der Übersicht halber, an den Anfang der Datei.

Unsere *kinoleak.py* sieht dann so aus:

Code:

```
from resources.lib.gui.gui import cGui from resources.lib.gui.guiElement
import cGuiElement from resources.lib.handler.requestHandler import
cRequestHandler from resources.lib.parser import cParser from
resources.lib.handler.ParameterHandler import ParameterHandler
SITE_IDENTIFIER = 'kinoleak' SITE_NAME = 'KinoLeak.tv' #SITE_ICON =
'kinoleak.png'  URL_MAIN = 'http://kinoleak.tv/'  def load():    pass
```

Als nächstes passen wir die Funktion load() folgendermaßen an:

Code:

```
def load():    oGui = cGui()        oGui.addFolder(cGuiElement('Neuste
Filme', SITE_IDENTIFIER, 'showNewMovies'))
oGui.addFolder(cGuiElement('Suche', SITE_IDENTIFIER, 'showSearch'))
oGui.setEndOfDirectory()
```

oGui ist im wesentlichen für die Darstellung des Menus verantwortlich.

Objekte vom Typ *cGuiElement* repräsentieren die Menueinträge.

Per *cGuiElement('Neuste Filme', SITE_IDENTIFIER, 'showNewMovies')* wird ein neues cGuiElement erzeugt, welches den Anzeigetitel "Neuste Filme" trägt und auf die Funktion *showNewMovies* in der *kinoleak.py*(SITE_IDENTIFIER) verweist.

Jedes fertige GuiElement **muss** per *addFolder()* an oGui übergeben werden um angezeigt zu werden.

Analog dazu wird ein zweiter Menueintrag mit dem Titel "Suche" erstellt, welcher auf die Funktion *showSearch* in unserer *kinoleak.py* verweist.

Mittels *oGui.setEndOfDirectory()* wird angezeigt, dass keine weiteren Einträge folgen. Diese Zeile ist für die Anzeige des Menus **erforderlich**.

Nach diesen Schritten sollten wir im xStream Hauptmenu einen Eintrag "KinoLeak.tv" vorfinden, welcher uns zu einem Menu mit den Einträgen "Neuste Filme" und "Suche" führt.

Beide Einträge sind bis jetzt natürlich ohne Funktion und sollten zu einem Fehler führen, da die entsprechenden Funktionen noch nicht in der *kinoleak.py* implementiert sind.

Als nächstes sollten wir unsere zwei Menueinträge ("Beuste Filme" und "Suche") mit Leben befüllen. Dafür benötigen wir, wie bereits erwähnt, einen Webbrowser mit Entwicklerwerkzeugen, welche eigentlich bei jedem üblichen Browser dabei sind, oder ein Browseraddon wie Firebug. Ich werde in diesem HowTo Firefox mit Firebug benutzen.

Filmauflistung:

Wenn wir mit unserem Browser unsere Beispielseite "kinoleak.tv" aufrufen sehen wir im mittleren Bereich den Abschnitt "New Movies". Die Filme die dort angezeigt werden, sollen auch unter unserem Menueintrag "Neuste Filme" aufgelistet werden.

Dazu müssen wir eine Funktion implementieren welche folgende "Aktionen" durchführt:

- url aufrufen, die den Teil "New Movies" enthält
- Informationen (Tiel, Link, Cover ...) für jeden Film aus dem Quelltext der Seite extrahieren
- extrahierte Informationen in *GuiElemente* übertragen und an ein *Gui*-Objekt übergeben

Die zu implementierende Funktion nennen wir *"showNewMovies*", da wir bereits vorher festgelegt hatten, dass der Menueintrag "Neuste Filme" eine Funktion mit diesem Namen aufruft. Der erste Teil der fertigen Funktion sieht aus wie folgt:

Code:

```
def showNewMovies():     oGui = cGui()      params = ParameterHandler()
oRequestHandler = cRequestHandler(URL_MAIN)    sHtmlContent =
oRequestHandler.request()
```

oGui kennen wir bereits aus dem vorherigen Abschnitt.
params enthält alle Parameter, die von vorherigen Menueinträgen übergeben wurden. Dazu kommen wir später nochmal.
Mit den letzten zwei Zeilen wird der Inhalt (html-Code) der url "http://kinoleak.tv" abgerufen und in der Variable *sHtmlContent* abgelegt

Aus dem html-Code werden nun die Filminformationen extrahiert. Der dazu nötige Teil der Funktion sieht so aus:
Code:

```
pattern = 'class="tabel-topasd".*?<a href="([^"]+)"><img src="([^"]+)"
title="([^"]+)".*?<span.*?>([^<>]+)</span>.*?title="([^"]+)"/>'    oParser
= cParser()    aResult = oParser.parse(sHtmlContent, pattern)    if not
aResult[0]:        return
```

pattern ist ein regulärer Ausdruck (RegEx), mittels *oParser.parse(sHtmlContent, pattern)* wird dieser auf den in *sHtmlContent* gespeicherten html-Code angewendet. Alle Treffer werden in *aResult* abgelegt.
Die if-Anweisung prüft ob *aResult[0] == false* ist, ist dies der Fall konnten keine Treffer gefunden werden und die Funktion wird mittels *return* beendet.

Ok, wie kommt man nun zu so einem RegEx?

Man öffnet die zu parsende Seite mit einem Webbrowser und startet die Entwicklerwerkzeuge, oder wie in meinem Fall Firebug.

Mit der Inspektionsfunktion von Firebug werden bei "mouseover" einzelne bzw. zusammengehörige Elemente eingefärbt und deren html-Code im unteren Teil des Bildschirms angezeigt (siehe Bild).

Somit lässt sich schon mal leicht bestimmen nach welchen Merkmalen man im Quelltext schauen muss wenn man bestimmte Informationen extrahieren möchte.

In diesem Fall sieht man, dass die relevanten Filme immer in einer *table* mit *class="tabel-topasd"* stecken.

Man sollte jedoch nicht direkt den in dieser Firebug Ansicht angezeigten Quelltext kopieren und daraus einen RegEx bauen, da dieser nicht immer 100% mit dem originalen Quelltext übereinstimmt.

Wir öffnen also den Quelltext der Seite (z.B. Rechtsklick -> Seitenquelltext anzeigen) und suchen nach *class="tabel-topasd"*.

So finden wir Codeteile, die aussehen wie der folgende:

PHP-Code:

```
<table width="438" border="0" cellspacing="0" cellpadding="0" class="tabel-
topasd" style="float:left;width:438px;margin-left:8px;margin-
bottom:5px;margin-top:2px;background-color: #181c20;">          <tr>
<td width="48" valign="top"><a href="index.php?site=Movies&id=287"><img
src="http://i.imglol.de/Scary-Movie-5-Poster9832z8.jpg" title="Scary Movie
5 (2013)" width="43px" height="60px"/></a></td>            <td
valign="top">              <span style="font-size:12px;color:#0080FF;font-
```

```
family:'verdana';"><b><u><a href="index.php?site=Movies&id=287"
title="Scary Movie 5 (2013)"style="color:#0080FF;text-decoration:
none;text-underline: none;">Scary Movie 5 (2013)</a></u></b></span><br />
<span style="font-size:11px;font-family:'verdana';" >Als die drei völlig
verwilderten Kinder von Dans Bruder in einer Waldhütte gefunden werden,
beschließen Jody (Ashley Tisdale) und Dan (Simon Rex),..</span>
</td>            <td width="30" align="right" valign="top">
<img src="img/787419hd720p.png" width="26px" title="HD 720p"/>
</td>            </tr>          </table>
```

Aus jedem Teil wollen wir Titel, Link, Cover, Plot und evtl. noch die Qualität extrahieren.

Hinweis:
es gibt online RegEx Tester (z.B. regexpal.com) damit kann man RegEx auf einen Text anwenden und muss das Ergebnis nicht ständig in python ausprobieren.

Der Anfang eines für uns relevanten Codeteils wird durch *class=tabel-topasd"* markiert, damit beginnt auch unser RegEx. Der Link steckt in *<a href="index.php?site=Movies&id=287">*.

Mit <a href="([^"]+)"> lässt sich daraus der relevante Teil *index.php?site=Movies&id=287* extrahieren. Die Klammern ( ) definieren eine Gruppe, dies ist üblicherweise ein Bereich der als Ergebnis zurückgegeben wird. [^"] bedeutet wir akzeptieren jedes Zeichen außer(^) " und zwar soviel wie möglich(+).

Zwischen unserem Regex Anfang und dem Link kann vieles verschiedenes Stehen das wollen wir nicht extra angeben und es ist auch unwichtig. Das können wir einfach per *.*?* lösen, dieser Ausdruck passt auf alles.

Unser RegEx sieht damit erst mal so aus: *class="tabel-topasd".*?<a href="([^"]+)">*
Das Cover und der Titel folgen direkt auf den Link in ähnlichen Anordnungen: *<img src="http://i.imglol.de/Scary-Movie-5-Poster9832z8.jpg" title="Scary Movie 5 (2013)"*.

Auch hier sind die relevanten Teile direkt mit " " umgeben so das wir mit *([^"]+)* die Gruppe definieren die wir extrahieren wollen.
Unser RegEx wird damit zu: *class="tabel-topasd".*?<a href="([^"]+)"><img src="([^"]+)" title="([^"]+)"*
Bist zum Plot kommt erst mal wieder einiges das nicht so wichtig ist da nutzen wir wieder *.*?* .

Der Plot selbst steckt dann in *Als die drei völlig verwilderten Kinder von Dans Bruder in einer Waldhütte gefunden werden, beschließen Jody (Ashley Tisdale) und Dan (Simon Rex),..*.

Dieses Mal ist der relevante Text nicht von " " sondern von > < eingeschlossen. Damit wir den span "erwischen" nehmen wir in den RegEx *<span* , dann *.*?* da uns die Formatierungen egal sind und *>([^<>]+)*; gibt zusammen *<span.*?>([^<>]+)*.

Dadurch dass wir < **und** > auschließen verhindern wir das wir den anderen Span erwischen, der nicht den Plot sonden nochmal Titel und Link enthält.

Unser RegEx sieht jetzt so aus: *class="tabel-topasd".*?<a href="([^"]+)"><img src="([^"]+)" title="([^"]+)".*?<span.*?>([^<>]+)*
Danach extrahieren wir noch den Teil in welchem die Auflösung stekt, *title="HD 720p"/>*.
Das geht wie bei den vorherigen Teilen.

Der fertige RegEx ist dann der, welcher auch im Code zu finden ist:
class="tabel-topasd".*?<a href="([^"]+)"><img src="([^"]+)" title="([^"]+)".*?<span.*?>([^<>]+).*?title="([^"]+)"/>

Nach dem Parsen befinden sich alle Treffer in *aResult[1]* und werden mit einer Schleife durchlaufen und in GuiElemente übertragen.
Der letze Teil der Funktion sieht wie folgt aus:

Code:

```
total = len(aResult[1]) # Anzahl der Treffer    for link, img, title, plot,
qual in aResult[1]:        titleParts = title.split('(') # Titel von Jahr
trennen        movieTitle = titleParts[0].strip().decode('iso-8859-
1').encode('utf-8') # encoding anpassen wegen Umlauten
guiElement = cGuiElement(movieTitle, SITE_IDENTIFIER, 'getHosters')
guiElement.setThumbnail(img) #Cover als Thumbnail setzen
guiElement.setDescription(plot.decode('iso-8859-1')) # Filmbeschreibung
setzen, decode wegen Umlauten        if len(titleParts)>1:            tag =
titleParts[-1].replace(')','')            if tag.isdigit() and len(tag)==4:
guiElement.setYear(tag)        guiElement.setMediaType('movie')        if
'720p' in qual: # erst mal unwichtig            guiElement._sQuality = 720
elif '1080p' in qual:            guiElement._sQuality = 1080
params.setParam('siteUrl',link)        oGui.addFolder(guiElement, params,
bIsFolder = False, iTotal = total)    oGui.setView('movies') #diese Liste
unterliegt den automatisch ViewSettings für Filmlisten
oGui.setEndOfDirectory()
```

link, img, title, plot, qual liefern den Inhalt der Gruppen aus dem Treffer bzw. dem RegEx. Die Reihenfolge entspricht der der Gruppen im RegEx.

Wir trennen den Titel vom Jahr damit es ordentlich ist und der metahandler keine Probleme bekommt.

Mit *cGuiElement(movieTitle, SITE_IDENTIFIER, 'getHosters')* erzeugen wir ein neues GuiElement für den jeweiligen Film, welches die Funktion 'getHosters' aufrufen wird, welche wir ebenfalls noch implementieren müssen.

Mittels *guiElement.setYear()* setzen wir das Jahr separat. Das ist besonders für den metahandler hilfreich das dieser so die Filme besser eindeutig identifizieren kann.

Per *guiElement.setMediaType('movie')* damit wird diese GuiElememnt als "movie" gekennzeichnet, dies ist zwar nur für den metahandler relevant, der korrekte MediaType sollte aber nach Möglichkeit immer gesetzet werden. Bei gesetztem MediaType und aktiviertem metahandler werden die Metainformationen automatisch geladen.

params.setParam('siteUrl',link) fügt den aktuellen Parametern den Parameter 'siteUrl' hinzu, oder aktualisiert diesen. Dieser Parameter enthält den Link bzw. den relevanten Teil des Links zur Seite des Films.

oGui.addFolder(guiElement, params, bIsFolder = False, iTotal = total) übergibt das GuiElement an das Gui, **inklusive** der zusätzlichen Parameter.

bIsFolder = False zeigt an das nun nur noch die Hoster folgen aber keine weitere Listen, xStream wird darauf (je nach Einstellungen) automatisch mit den Playroutinen reagieren und dem GuiElement entsprechende Contextmenü Einträge hinzufügen. **ERFORDERLICH**
- *iTotal = total* bewirkt, dass beim laden der Liste ein Ladebalken angezeigt werden kann

Die komplette Funktion **showNewMovies()**:
Code:

```
def showNewMovies():     oGui = cGui()      params = ParameterHandler()
oRequestHandler = cRequestHandler(URL_MAIN)    sHtmlContent =
oRequestHandler.request()    # parse movie entries    pattern =
'class="tabel-topasd".*?<a href="([^"]+)"><img src="([^"]+)"
title="([^"]+)".*?<span.*?>([^<>]+)</span>.*?title="([^"]+)"/>'    oParser
= cParser()    aResult = oParser.parse(sHtmlContent, pattern)    if not
aResult[0]:        return    total = len(aResult[1]) # Anzahl der Treffer
for link, img, title, plot, qual in aResult[1]:        titleParts =
title.split('(') # Titel von Jahr trennen        movieTitle =
titleParts[0].strip().decode('iso-8859-1').encode('utf-8') # encoding
anpassen wegen Umlauten                  guiElement =
cGuiElement(movieTitle, SITE_IDENTIFIER, 'getHosters')
guiElement.setThumbnail(img) #Cover als Thumbnail setzen
guiElement.setDescription(plot.decode('iso-8859-1')) # Filmbeschreibung
setzen, decode wegen Umlauten        if len(titleParts)>1:            tag =
titleParts[-1].replace(')','')            if tag.isdigit() and len(tag)==4:
guiElement.setYear(tag)        guiElement.setMediaType('movie')        if
'720p' in qual: # erst mal unwichtig            guiElement._sQuality = 720
elif '1080p' in qual:            guiElement._sQuality = 1080
params.setParam('siteUrl',link)        oGui.addFolder(guiElement, params,
bIsFolder = False, iTotal = total)    oGui.setView('movies') #diese Liste
unterliegt den automatisch ViewSettings für Filmlisten
oGui.setEndOfDirectory()
```

Nach diesem Schritt sollten wir, nachdem wir den Menupunkt 'Neuste Filme' gewählt haben, eine Liste den neuesten Filmen unserer Beispielseite erhalten.

Da jeder Film in dieser Liste auf die nicht implementierte Funktion 'getHosters' verweist, wird zum jetzigen Zeitpunkt die Auswahl eines Films aus dieser Liste wieder zu einem Fehler führen.

Als nächsten werden wir also die Funktion getHosters implementieren.

Hosterliste:

Die Funktion *getHosters* für unsere Beispielseite ruft die Seite des jeweiligen Films auf und gibt die für diesen Film verfügbaren Streamhoster und deren Links zurück.
Wir öffnen die Seite von einem Film im Webrowser und versuchen die relevanten Stellen wieder im Quelltext wiederzufinden.
Die Links zu den Hostern befinden sich in den iframes mit den Großbuchstaben.

Funktion **getHosters()**:
Code:

```
def getHosters():    oParams = ParameterHandler() #Parameter laden    sUrl
= oParams.getValue('siteUrl')  # Weitergegebenen Urlteil aus den Parametern
holen         oRequestHandler = cRequestHandler(URL_MAIN+sUrl) # gesamte
Url zusammesetzen    sHtmlContent = oRequestHandler.request()         #
Seite abrufen     sPattern = 'IFRAME SRC="([^"]+)"'    oParser = cParser()
aResult = oParser.parse(sHtmlContent, sPattern)    hosters = []
# hosterliste initialisieren    sFunction='getHosterUrl'
# folgeFunktion festlegen     if aResult[0]:        for aEntry in
aResult[1]:            hoster = {}            hoster['link'] = aEntry
# extract domain name            temp = aEntry.split('//')            temp
= str(temp[-1]).split('/')            temp = str(temp[0]).split('.')
hoster['name'] = temp[-2]            hosters.append(hoster)
hosters.append(sFunction)    return hosters
```

Die Funktion gibt eine Liste aller für den jeweiligen Film verfügbaren Hoster zurück.

Jeder Hostereintrag muss zwei Felder enthalten:
- '*link*' mit dem weiterführenden Link
- '*name*' mit dem Domainnamen des Hosters

Am Ende der Liste wird der Name der nachfolgen Funktion angefügt. Das ist notwendig da nicht bei jeder Seite an dieser Stelle bereits die direkten Links zu den Hostern verfügbar sind.

Bei unserer Beispielseite sind die Hosterlinks für jeden verfügbare jedoch bereits jetzt vorhanden, und werden durch das parsen mit dem *sPattern* aus dem Quellcode der Seite extrahiert.

Die nachfolgende Funktion *getHosterUrl* ist somit, für unser Beispiel, äußerst simpel zu implementieren.

Funktion **getHosterUrl()**:
Code:

```
def getHosterUrl(sStreamUrl = False):

     if not sStreamUrl:
         params = ParameterHandler()
         sStreamUrl = oParams.getValue('url')
     results = []
     result = {}
     result['streamUrl'] = sStreamUrl
     result['resolved'] = False
     results.append(result)
     return results
```

Diese Funktion muss über einen optionalen Parameter verfügen an welchen eine Url übergeben werden kann.

Der übergebene Wert entspricht dem Wert des zum Hoster gehörigen Feld '*link*' welcher in der Funktion *getHosters()* gesetzt wird.

Um auch zu funktionieren, wenn die Einstellung "zeige Hosterliste als Verzeichnis" wird geprüft ob der übergebene Parameter seinem default Wert entspricht.

Dies übernimmt der if-Block.
Ist das der Fall wird aus dem ParameterHandler der Wert des Parameters *'url'* gelesen.
Auch dieser entspricht dem Wert des zum Hoster gehörigen Feld '*link*' welcher in der Funktion *getHosters()* gesetzt wird.

Diese Funktion dient dazu den Link bzw. die Links zum Stream des jeweiligen Hosters zu liefern. Jedoch nicht unbedingt den Link zur Datei, welche tatsächlich gestream wird.

Jeder Link wird in einem dictionary (*result*) angelegt welches folgende Felder unterstützt:
- *streamUrl* : der Link zum Stream
- *resolved* : False, wenn der Link noch z.B. vom urlresolver aufgelöst werden muss; True, falls es sich bereits um einen direkt abspielbaren Link handelt
- *title* : ein alternativer Titel z.B. falls der Film in mehere Teile gesplitted wurde kann hier der Titel mit nummer angegeben werden.

Jedes dictionary wird in einer Liste (*results*) abgelegt, welche der Rückgabewert dieser Funktion ist. Die Liste wird verwendet um Filme zu unterstützen, welche in mehere Teile gesplitted.

Bei unserer Beispielseite ist die jedoch nicht der Fall

Wie man dem Code entnehmen kann muss in unserem Beispiel der Wert des Felds 'link' aus der vorherigen Funktion einfach nur unverändert durchgereicht werden.

Hinweis:
Ist der Wert Feld '*resolved* ' *False* wird automatisch das externe Module *urlresolver* verwendet, um für die übergeben Url den tatsächlichen Link zur streambaren Datei zu finden.

Dafür muss für den entsprechenden Hoster jedoch auch ein funktionierender Resolver im urlresolver vorhanden sein.

Für einen Hoster unserer Beispielseite ist jedoch kein Resolver vorhanden.
Darum könnte man sich auch innerhalb des Site-Plugins kümmern.

Die sauberere Variante wäre jedoch einen extra Resolver zu schreiben (http://t0mm0.github.io/xbmc-urlresolver/...index.html).

Ok, das wars erst mal.
Wer den Text gelesen hat sollte mit dem vorgestellten Code der einzelnen Funktionen eine funktionierende kinoleak.py erstellen können