(typo3-tutorials/extensions/scheduler.html)

eigene Tasks erstellen

(index.php?eID=tx_cms_showpic&file=uploads%2Fpics%2Fscheduler-liste.png&md5=285d4b8475e9a7c4ad2ecf00ddbeedbc2ad29248&parameters[0]=YTo0OntzOjU6IndpZHRoIjtzOjM6IjgwMCI7czo2OiJoZWlnaHQiO3M6NDoiNjAw&parameters[1]=bSI7czo3OiJib2R5VGFnIjtzOjQxOiI8Ym9keSBzdHlsZT0ibWFyZ2luOjA7IGJh&parameters[2]=Y2tncm91bmQ6I2ZmZjsiPiI7czo0OiJ3cmFwIjtzOjM3OiI8YSBocmVmPSJqYXZh&parameters[3]=c2NyaXB0OmNsb3NlKCk7Ij4gfCA8L2E%2BIjt9)

Um eigene Tasks zu erstellen müssen wir auf jeden Fall eine eigene Extension erzeugen. Der kickstarter wäre hierfür der richtige Ansprechpartner. Wir brauchen kein Plugin, keine Datenbank oder sonstigen zusätzlichen Felder. Ein paar Daten bei "general informations" eintragen und die Extension installieren/aktivieren->Fertig.

Die fertige Extension runterladen (fileadmin/templates/downloads/T3X_sfpinger-0_0_0-z-201002051443.t3x)

Wir legen nun die Datei ext_localconf.php im Rootverzeichnis der Extension an und geben der scheduler-Extension bescheid, dass sich auch in unserer Extension Tasks befind

```
<?php
if (!defined ('TYPO3_MODE')) die ('Access denied.');

$GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['scheduler']['tasks']['tx_sfpinger_pinger'] = array(
    'extension'        => $_EXTKEY,
    'title'            => 'Server anpingen',
    'description'      => 'Dieser Task benötigt eine IP-Adresse, die er alle 15 Minuten anpinge
);
?>
```

Allein durch diese Angaben, könnt Ihr Euren neuen Task schon im "Planer" auswählen und konfigurieren. Allerdings ist unser Task noch ohne Funktion! "sfpinger" habe ich fett geschrieben, weil dass die Klasse ist, in der die weitere Funktionalität definiert wird. Dazu legen wir in unserer Extension einen neuen Ordner "tasks" an und erstellen in diesem Ordner eine Datei mit dem Namen class.tx_sfpinger_pinger.php. Die Datei muss nicht so heißen, aber laut Codingguideline von TYPO3 sollte sie so heißen. Jeder Task MUSS von der Klasse tx_scheduler_Task erben und mindestens die Methode "execute()" beinhalten. Die Datei könnte also wie folgt aussehen:

```
<?php
class tx_sfpinger_pinger extends tx_scheduler_Task {
        public function execute() {
                $fp = @fsockopen('typo3.sfroemken.de', 80, $errno, $errstr, 15);
                if(!$fp) {
                        mail('meine@mailadresse.de', 'Server Error', 'Fehler: '.$errstr.'<br />
                        return false;
                } else return true;
        }
}
?>
```

Das @-Zeichen vor der fsockopen-Funktion ist dafür da, dass Ihr nicht die PHP-Warnungen seht, wenn Ihr den Task per Hand ausführt und der Server nicht erreichbar sein sollte.

Als letzten Schritt müssen wir dem scheduler noch sagen WO er die Klasse tx_sfpinger_pinger finden kann. Seit TYPO3 4.3.0 kann das mit dem Autoloader realisiert werden. Legt dazu eine ext_autoload.php in das Rootverzeichnis Eurer Extension an und füllt sie mit folgendem Inhalt:

```
<?php
return array(
    'tx_sfpinger_pinger' => t3lib_extMgm::extPath('sfpinger', 'tasks/class.tx_sfpinger_pinger.p
);
?>
```

Nun könnt Ihr einen Task anlegen und werden nur dann informiert, wenn der Port 80 Eures Servers nicht erreichbar ist. Wenn Ihr keine Mail bekommt ist alles in Ordnung. Tragt mal statt Serverport 80 den Port 123 ein und versucht es nochmal den Task zu starten. Ich hab meine Mail mit den Fehlermeldungen bekommen...

Dynamische IP-Adresse und Port

Wir wollen nun zwei weitere Felder in unseren Task integrieren in denen der User eine eigene IP-Adresse bzw. Webseite und einen Port eintragen kann. Wer will kann sich dazu mal die Funktion getRegisteredClasses() in der /mod1/index.php der scheduler-Extension anschauen. Dort sehen wir, dass es noch einen 4ten Parameter für die Funktion gibt, die uns das Hinzufügen von weiteren Feldern erlaubt. Wir ändern unsere ext_localconf.php entsprechend ab:

```
<?php
if (!defined ('TYPO3_MODE')) die ('Access denied.');

$GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['scheduler']['tasks']['tx_sfpinger_pinger'] = array(
    'extension'        => $_EXTKEY,
    'title'            => 'Server anpingen',
    'description'      => 'Dieser Task benötigt eine IP-Adresse, die er alle 15 Minuten anpinge
    'additionalFields' => 'tx_sfpinger_pinger_addFields'
);
?>
```

Wie Ihr ahnen könnt werden weitere Felder auch wieder über eine Klasse definiert. Also ab in den tasks-Ordner und dort die Datei tx_sfpinger_pinger_addFields anlegen mit folgendem Inhalt:

```
<?php
class tx_sfpinger_pinger_addFields implements tx_scheduler_AdditionalFieldProvider {
    public function getAdditionalFields(array &$taskInfo, $task, tx_scheduler_Module $parentObj

        if (empty($taskInfo['ip'])) {
            if($parentObject->CMD == 'edit') {
                $taskInfo['ip'] = $task->ip;
            } else {
                $taskInfo['ip'] = '';
            }
        }

        if (empty($taskInfo['port'])) {
            if($parentObject->CMD == 'add') {
                $taskInfo['port'] = '80';
            } elseif($parentObject->CMD == 'edit') {
                $taskInfo['port'] = $task->port;
            } else {
                $taskInfo['port'] = '';
            }
        }

        // Write the code for the field
        $fieldID = 'task_ip';
        $fieldCode = '<input type="text" name="tx_scheduler[ip]" id="' . $fieldID . '" value="
        $additionalFields = array();
        $additionalFields[$fieldID] = array(
            'code'     => $fieldCode,
            'label'    => 'IP-Adresse/Webseite'
        );

        // Write the code for the field
        $fieldID = 'task_port';
        $fieldCode = '<input type="text" name="tx_scheduler[port]" id="' . $fieldID . '" value=
        $additionalFields[$fieldID] = array(
            'code'     => $fieldCode,
            'label'    => 'Port'
        );

        return $additionalFields;
    }

    public function validateAdditionalFields(array &$submittedData, tx_scheduler_Module $parent
        $submittedData['ip'] = trim($submittedData['ip']);
        $submittedData['port'] = trim($submittedData['port']);
        return true;
    }

    public function saveAdditionalFields(array $submittedData, tx_scheduler_Task $task) {
        $task->ip = $submittedData['ip'];
        $task->port = $submittedData['port'];
    }
}
?>
```

getAdditionalFields

Die beiden ersten empty-ifs sollen auf jeden Fall drin bleiben. Wenn Ihr sie weglasst, werden Eure Daten, die Ihr in die Felder eintragen wollt nicht gespeichert und/oder beim Bearbeiten nicht erneut angezeigt. Die beiden mitleren Abschnitte erstellen die neuen Felder. Hier ist wichtig, dass Ihr tx_scheduler als Namen angebt und nicht den Namen Eurer eigenen Extension. Das additionalFields-Array kann neben code und label auch noch Informationen für die csh-Hilfe enthalten. Schaut Euch dazu die Beispiele in der scheduler-Extension an.

validateAdditionalFields

Hier können Eure eingetragenen Werte überprüft werden. In meinem Fall nutze ich diese Funktion, um überflüssige Leerzeichen am Anfang und am Ende der Werte zu entfernen. Wenn diese Funktion TRUE zurückgibt wird die

Funktion saveAdditionalFields ausgeführt.

saveAdditionalFields

Alle eingegebenen Werte werden hier nun dem scheduler-Task-Objekt hinzugefügt. Heißt in unserer Datei tx_sfpinger_pinger.php können wir nun mit $this->ip und $this->port auf die abgespeicherten Werte zugreifen. Schauen wir uns die neue tx_sfpinger_pinger.php mal an:

```
<?php
class tx_sfpinger_pinger extends tx_scheduler_Task {
    public function execute() {
        $fp = @fsockopen($this->ip, $this->port, $errno, $errstr, 15);
        if(!$fp) {
            mail('firma@sfroemken.de', 'Server Error', 'Fehler: '.$errstr.CHR(10).
                'Fehlernummer: '.$errno.CHR(10).
                'Server: '.$this->ip.CHR(10).
                'Port: '.$this->port);
        }
        return true;
    }
    public function getAdditionalInformation() {
        return 'Server: '.$this->ip.' wird auf Port: '.$this->port.' gescannt.';
    }
}
?>
```

Hier seht Ihr nun, dass ich den Fehlerbericht, der per Mail versendet werden soll noch um den Server und Port ausgeweitet habe. Außerdem besitzt unsere Klasse nun noch die Methode getAdditionalInformations. Diese Methode hat allerdings nix mit "hat geklappt" und "hat nicht geklappt" zu tun, sondern eher damit, dass Ihr gleiche Tasks besser unterscheiden könnt. Legt mal mehrere unserer Tasks an und Ihr werdet feststellen, dass sie alle gleich aussehen. Erst Dank dieser Funktion erscheinen zusätzliche Informationen in der Liste, die Euch helfen die Tasks zu unterscheiden. In meinem Fall habe ich mich für den Servernamen und den Port entschieden.

Damit unsere Klasse wieder gefunden wird, müssen wir diese auch noch mit in die ext_autoload.php einbinden:

```
<?php
return array(
    'tx_sfpinger_pinger' => t3lib_extMgm::extPath('sfpinger', 'tasks/class.tx_sfpinger_pinger.p
    'tx_sfpinger_pinger_addfields' => t3lib_extMgm::extPath('sfpinger', 'tasks/class.tx_sfpinge
);
?>
```

Parallel oder nicht?

Bei jedem Task habt Ihr die Möglichkeit zwischen parallel oder nicht zu wählen. Soll heißen: Darf der aktuelle Task parallel zu anderen gleichen Tasks laufen oder nicht. Wenn Ihr in unserem Beispiel nur 3-5 Tasks angelegt habt, dann könnt Ihr bedenkenlos "parallel" wählen. Es werden dann dementsprechend bis zu 5 gleichzeitig laufende Dienste auf dem Server gestartet. Bei mehreren Hundert dieser Tasks würde ich von parallel allerdings abraten. 100 neue Tasks auf dem Server kann je nach Task den Server echt in die Knie zwingen. Hier wählt als bitte nicht parallel. Die Tasks werden dann hintereinander abgearbeitet