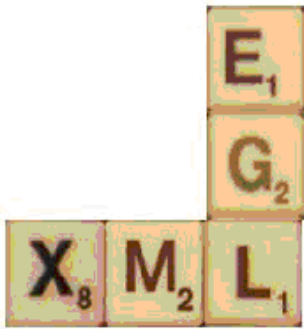


Scott Pecnik:

XML-Dokumente mit EGL und XPath parsen und abfragen



In der sich ständig entwickelnden Welt der Unternehmen ist XML zu einem Eckpfeiler des Austauschs von Informationen über das Internet bzw. zwischen Software verschiedener Art geworden. Auf die eine oder andere Weise nutzen alle neuen Anwendungen XML. Mit XML zu arbeiten ist jedoch nicht immer ganz einfach. Wenn Sie manuell einen Code zum Parsen von XML-Dokumenten schreiben wollen, muss dieser aus einer komplexen Logik mit einem endlichen Automaten in Verbindung mit einem nicht trivialen Parser für Zeichenfolgen bestehen. Glücklicherweise gibt es eine gute Alternative, mit der sich XML-Dateien benutzerfreundlich und ohne eine Zeile selbst geschriebenen Code abfragen lassen. Diese "Alternative" könnte schon ziemlich bald zum Standard für XML-Abfragen werden.

Sie wollen wissen, wie diese wunderbare Technologie heißt? Es ist XPath, die Antwort auf alle unsere Sorgen im Zusammenhang mit XML-Abfragen. Und ja, Sie haben es erraten: Wir haben die Kraft von XPath erfolgreich in EGL integriert. Mit der Hilfe einer einfachen EGL-Bibliothek können die Ergebnisse einer XML-Abfrage mit nur zwei Codezeilen sicher in einem String-Array gespeichert werden. XPath gibt Ihnen eine wirklich brauchbare API an die Hand, so dass Sie sich nicht mehr um das Was, Wie und Warum im Zusammenhang mit dem DOM (Document Object Model) kümmern müssen. (Mit der Erwähnung des DOM wollte ich jetzt niemandem einen Schrecken einjagen. Ich weiß natürlich, dass Sie auch ohne XPath bestens damit klarkommen!)

Alles was Sie brauchen, um Ihr EGL-Projekt mit XPath-Funktionalität auszustatten, sind zwei EGL-Dateien. Die eine Datei enthält ExternalTypes zur Überbrückung der Lücke zwischen Java und EGL. Die andere enthält Bibliotheksfunktionen, hinter denen sich die komplizierteren Schritte für die vollständige Implementierung verbergen.

Hier (siehe folgende Abbildung) ist ein Teil des ExternalType-Abschnittes, den Sie zu Ihrem EGL-Projekt hinzufügen. Er definiert eine Java-Klasse und deren Inhalt, so dass Sie diese über EGL aufrufen können. Die Definition der ExternalTypes für den Zugriff auf die XPath Java-Methoden hat Ihnen schon das ECO System Team abgenommen. [\[Anmerkung des Herausgebers – Um den vollständigen Quellcode hierfür zu bekommen, senden Sie eine Mail an Scott Pecnik \(pecnik@us.ibm.com \) oder Jon Sayles – \(jsayles@us.ibm.com\) – oder nehmen Sie an einem EGL-Workshop teil.\]](#)

XPath_Component.egl

package XPath;

ExternalType DocumentBuilderFactory **type** JavaObject
{JavaName = "DocumentBuilderFactory", PackageName = "javax.xml.parsers"}

static function newInstance() **returns** (DocumentBuilderFactory);

function setNamespaceAware(answer **boolean** in);

function newDocumentBuilder() **returns** (DocumentBuilder);

End

ExternalType DocumentBuilder **type** JavaObject
{JavaName = "DocumentBuilder", PackageName = "javax.xml.parsers"}

function parse(uri **string** in) **returns** (Document);

end

ExternalType Document **extends** JavaObj **type** JavaObject
{JavaName = "Document", PackageName = "org.w3c.dom"}

end

ExternalType JavaObj **type** JavaObject
{JavaName = "Object", PackageName = "java.lang"}

Die Bibliothek enthält zwei Funktionen (siehe unten). Eine Funktion wird aufgerufen, um der Anwendung mitzuteilen, wo im Dateisystem sich die XML-Datei befindet. Die andere ruft den XML-Parser von XPath auf.

```

/*
 * Finds the XML file on the filesystem
 * and prepares it to be read.
 *
 */
function setLocation(str String)
    domFactory DocumentBuilderFactory = DocumentBuilderFactory.new
    domFactory.setNamespaceAware(true); // never forget this!
    builder DocumentBuilder = domFactory.newDocumentBuilder();
    doc = builder.parse(str);
end

/*
 * Run a query on XML file and return
 * all results in an array.
 * Array is composed of type answer records
 */
function query(ans string[] inout, query String in)
    factory XPathFactory = XPathFactory.newInstance();
    xpath XPath = factory.newXPath();
    expr XPathExpression = xpath.compile(query);

    result NodeList = expr.evaluate(doc, XPathConstants.NODESET);
    temp Node;
    tempRec string;
    counter int;
    for (counter from 0 to result.getLength() - 1 by 1)
        temp = result.item(counter);
        tempRec = temp.getNodeValue();
        ans.appendElement(tempRec);
    end
end

```

Die oben aufgeführte Funktion (query) akzeptiert ein String-Array als Parameter, welches sie auffüllt und mit dem Ergebnis der Abfrage zurück gibt. Die Abfrage wird über eine EGL-Buchstabenzeichenfolge an die Funktion übergeben. Die Funktion übernimmt die zugrunde liegende Implementierung für Sie, so dass Sie mit EGL so produktiv wie möglich sein können.

Probieren Sie dies gleich einmal mit einer einfachen XPath-Anwendung aus! Angenommen, Sie speichern die Katalogdaten für eine Bibliothek mit Hilfe von XML. Der Einfachheit halber verwenden wir nur eine XML-Datei mit einer Tiefe auf Stammebene von drei. Die höchste Ebene ist der Typ "catalog", der zahlreiche Elemente des Typs "book" enthält. Jedes book-Element enthält mehrere untergeordnete Elemente, die dieses beschreiben.

```

<?xml version="1.0" ?>
- <catalog>
- <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with XML.</description>
</book>

```

Die XPath-Syntax enthält 13 verschiedene Operatoren. Diese Operatoren werden zur Erstellung einer erfolgreichen Abfrage miteinander kombiniert. Keine Sorge: XPath ist eine Abfragesprache, die man in ca. 20 Minuten erlernen kann. Hier ist ein Beispiel für eine XPath-Abfrage, die alle Buchtitel zurückgibt, deren Autor Kim Ralls ist:

```
//book[author='Ralls, Kim']/title/text()
```

In dieser Abfrage verwenden wir den Operator für rekursive Nachfolger (//). Hiermit wird der Parser angewiesen, die untergeordneten Elemente der XML-Datei rekursiv abwärts zu durchsuchen, bis das gesuchte Element, in diesem Fall "book", gefunden wird. Alternativ hierzu hätte man die Abfrage auch in dieser Form schreiben können:

```
/catalog/book[author='Ralls, Kim']/title/text()
```

Es ist offensichtlich, wie praktisch dieser Operator sein kann. Während Operatoren wie dieser lediglich zur Erhöhung der Produktivität existieren, gibt es auch ein paar hoch funktionale Operatoren wie zum Beispiel den Attribut-Operator ([@]). Durch die Verwendung von eckigen Klammern in der Abfrage können wir unsere Suche so filtern, dass nur Bücher mit dem Autor "Kim Ralls" betrachtet werden. Zuletzt geben wir an, dass wir nach dem Titel suchen, und dass dieser als Text ausgegeben werden soll. Einfach, oder?

Aber sehen Sie jetzt, wie leistungsstark XPath tatsächlich ist. Mit einer einzelnen Zeile schaffen Sie hier, wozu Sie in anderen Abfragesprachen hunderte gebraucht hätten. Nehmen wir die folgende Abfrage:

```
//book[@id='bk105']/description/text()
```

Auch diese Abfrage verwendet wieder den Operator für rekursive Nachfolger sowie den Attribut-Operator. Diesmal soll die Beschreibung aller Bücher zurück gegeben werden, deren ID gleich "bk105" ist.

Wenn Ihre EGL-Anwendung XML nutzt, empfehle ich Ihnen wärmstens, für alles, was mit Suchen zu tun hat, XPath zu verwenden. XPath kann Ihre Anwendung erheblich vereinfachen und die Produktivität der Programmierer wesentlich steigern.



Scott Pecnik ist Mitglied des TechEcosystem-Teams und Mitarbeiter des Research Triangle Park North Carolina. Während der vergangenen 5 Monate hat er an der Entwicklung und Implementierung von Studienmaterialien für Rational Business Developer and Enterprise Generation Language (EGL) mitgearbeitet. Er arbeitet eng mit Rational-Kunden zusammen und ist für seine kooperative und unterstützende Haltung bekannt. Nicht nur bei IBM ist Scott äußerst aktiv - er studiert gegenwärtig an der NC State University Informatik und Betriebswirtschaftslehre/Finanzen als Hauptfächer.

Wenn Sie mehr über die Verwendung von XPath wissen möchten, schreiben Sie unter der Adresse specnik@us.ibm.com an Scott Pecnik