Hycalper
- automatische Code Generierung -

Haymo Kutschbach, idalab GmbH

27. 03. 2006

quick start guide

Inhaltsverzeichnis

1 Funktionsweise **3**

1.1 Einbindung in Projektstruktur 3
1.2 Aufruf, Parameter 3
1.3 XML Typdefinitionen 3
1.3.1 gloabale- , lokale Typedefinitionen 5
1.3.2 Pfadangaben 5
1.4 Loops 6
1.4.1 Identifikation lokaler Loops 6
1.4.2 Import aus Datei 6
1.4.3 Pattern aus Loop ausnehmen 7
1.5 Enumerations 7
1.6 Ausgabe ins selbe File 7
1.7 vordefinierte Attribute 8
1.7.1 after 8
1.7.2 here 8
1.7.3 nextline 8
1.7.4 enregion 8
1.8 Spezialzeichen 8

2 Settings **8**

2.1 feste Einstellungen 8
2.1.1 LOOPSTART 9
2.1.2 LOOPEND 9
2.1.3 TYPELIST 9
2.1.4 ENUMPATTERN 9
2.1.5 include secure 9
2.1.6 parenthesize 9
2.1.7 comment 9
2.1.8 remove comments 9
2.1.9 keep tags 10

3 Tips'n Tricks **10**

1 Funktionsweise

Hycalper dient zum automatischen Duplizieren und Vervielfältigen von Sourcecode. Es ist im Rahmen der ILLibrary der idalab GmbH entstanden. Die Verwendung ist ausschließlich intern erlaubt! (Wer weiß, wem das gefällt... ⌣)
Hycalper ist ein Kommandozeilentool. Es liest die Anweisungen, welcher Kode wie repliziert wird aus Quelldateien („*templates*“). Es ist so konzipiert, daß diese Templates selbst vollwertigen Kode darstellen. Dadurch ist es möglich, ein File (z.B. C# Klasse) zuerst in einer einfachen Ausfertigung zu erstellen, zu kompilieren und zu debuggen, um es erst dann direkt als Ausgangspunkt für ein Template zu verwenden. Das Template bleibt dabei stets kompilierbar. Anhand von Tags innerhalb des Templates werden die Regeln und Bereiche festgelegt, die Hycalper zum Replizieren verwendet.

1.1 Einbindung in Projektstruktur

Ein template stellt eine Anweisung dar, wie **ein** einzelnes Output file erzeugt werden soll. Dabei werden genau definierbare Bereiche innerhalb des Templates extrahiert und anhand flexibler Ersetzungsregeln in eine neue Datei geschrieben. Diese Ausgabe sollte als Source Kode im Projekt verwendet werden. Alternativ kann auch das template selbst als Ausgabe verwendet werden. In diesem Fall ist natürlich das Template selbst auch in die Projektstruktur eingebunden. Letzterer Fall zeigt vor allem den Vorteil, daß der Sourcecode direkt im Template kompiliert und debugged werden kann.

1.2 Aufruf, Parameter

Hycalper hat 2 Kommandozeilenparamter:

1. Der Pfad, in welchem nach Source Files gesucht wird. Innerhalb des Verzeichnisses werden alle Dateien mit der Endung **.cs* gesucht und bearbeitet. Wird Hycalper ohne Parameter aufgerufen, wird im aktuellen Verzeichnis der exe-Datei gesucht. (Die Extension kann im Source Kode geändert werden.)

2. wird Hycalper mit einem 2. Parameter aufgerufen (inhalt ist egal), wartet Hycalper nicht auf eine Usereingabe, nachdem alle Files bearbeitet wurden, und beendet sich stillschweigend selbst.

1.3 XML Typdefinitionen

Jedes Template beinhaltet (an *fast* beliebiger Stelle) einen oder mehrere XML Bereiche, in welchen alle Informationen kodiert sind, die Hycalper benötigt. Jeder XML Bereich sollte in weitere (Block-)Kommentare eingeschlossen werden, um ein Aufbrechen der Programmsyntax zu vermeiden.

Das Root Tag des XML Bereiches wird mittels des Strings „!HC:TYPELIST:“ lokalisiert. Als Root Tag erwartet Hycalper das nächste Vorkommen von $\langle hycalper \rangle$. Das identifizierende Tag $!HC : TYPELIST :$ kann mit dem internen Settings 2.1.3 verändert werden. Die Informationen im XML Abschnitt enthalten:

- Angaben zum Outputfile. Die Angabe erfolgt entweder als absoluter oder als relativer Pfad. Relative Pfade sollten bevorzugt werden. Der Filename ist ein Tag direkt unterhalb des Root Tags. Es heißt „outfile“. Ein Bsp. könnte so aussehen:

  $$\langle outfile \rangle .. \max .cs \langle / outfile \rangle$$

  Der output-file tag ist optional. Nur im ersten XML Abschnitt eines templates wird er überhaupt berücksichtigt. Findet sich im ersten Hycalper XML Abschnitt eines Templates *kein* $\langle outfile \rangle$ tag, so wird das aktuelle template file selbst modifiziert und unter selben Namen gespeichert. Details -> siehe 1.6.

- beliebige *Typ Definitionen mit Ersetzungs-Attributen* (Anweisungen). Ebenfalls auf oberster Ebene im XML befinden sich beliebig viele Typ-Definitionen. Dabei verweist ein Typ auf alle Vorkommen des Tags innerhalb eines bzw. aller Kode-Loops (1.4) einer Datei. Jedes Vorkommen wird dabei sukzessive mit allen Ersetzungen des Typs ersetzt. Die Anzahl der Ersetzungen ist beliebig. Allerdings müssen alle Typdefinitionen die selbe Anzahl von Ersetzungen haben. Hycalper erstellt für jede Ersetzung aller Typen eine separate Loop-Kopie, in der jeweils die Vorkommen des Suchstrings des Typs mit der Ersetzung an der entsprechenden Stelle ersetzt wurden.

Ein **Beispiel** für eine XML Typdefinition, welche mit C#-Kommentar in ein *template* eingebettet wurde:

```
/*!HC:TYPELIST:
<hycalper>
<outfile>..\max.cs</outfile>
<type>
    <source locate="after">
        TinCls
    </source>
    <destination>complex</destination>
    <destination>float</destination>
    <destination>fcomplex</destination>
</type>
<type>
    <source locate="after">
```

```
        ToutCls
    </source>
    <destination>double</destination>
    <destination>float</destination>
    <destination>float</destination>
</type>
</hycalper>
*/
```

Hier werden 2 Typen definiert: `TinCls` und `ToutCls`. Jeder Typ hat 3 Ersetzungen. Diese werden in `<destination>` Tags eingeschlossen. Findet Hycalper nun im weiteren Text des Files in einem Loop Bereich (siehe 1.4) Vorkommen von `TinCls` oder `ToutCls`, werden unter dem Loop 3 Kopien desselben Loops erstellt, in welchen die entsprechenden Vorkommen ersetzt wurden. Dabei ersetzt Hycalper in der ersten Loop Kopie alle Vorkommen von `TinCls` mit `complex`, in der zweiten Kopie mit `float` usw., und alle Vorkommen von `ToutCls` in der ersten Kopie mit `double`, in der zweiten Kopie mit `float` usw. Nach Abarbeiten aller Loops, wird der Output im File max.cs im übergeordneten Ordner gespeichert. Die Tags innerhalb der `source` Abschnitte sind beliebig wählbar.

1.3.1 gloabale- , lokale Typedefinitionen

Der erste XML Typedefinitionsabschnitt in einem Template definiert eine globale Definition von Ersetzungsregeln. Diese wird für alle Loops und für alle Enums verwendet. Es ist zusätzlich möglich, für einen Loop diese Definition (quasi lokal) zu überschreiben. Dazu kann innerhalb der einschließenden Loop Markierungen (i.d.R. 'region HYCALPER LOOPSTART' und 'endregion HYCALPER LOOPEND', s.u.) ein neuer XML Abschnitt definiert werden. Dieser muß genau den selben Regel folgen, wie der globale Abschnitt (s.o.). Es ist ohne weiteres möglich, komplett andere Tags und/oder Pattern zu verwenden. Auch die Anzahl der Pattern darf von der globalen Definition abweichen. Dieses lokale Pattern ist nur innerhalb des aktuellen Loops gültig. Nach Abarbeiten des Loops wird wieder der gloabale XML Abschnitt bzw. ein neuer lokale XML Abschnitt (falls vorhanden) verwandt.

1.3.2 Pfadangaben

Als Pfadangaben sind relative und absolute Pfade zulässig. Relative Pfade sollten bevorzugt verwandt werden und sind dabei relativ zum Pfad des aktuellen Templates zu verstehen.

1.4 Loops

Zur Identifikation der Loop Bereiche innerhalb des Source Kodes werden vordefinierte Strings verwendet. Per Default lauten diese Strings:

- Loopanfang: #region HYCALPER LOOPSTART
- Loopende: #endregion HYCALPER LOOPEND

Diese Strings können im Source Kode geändert werden (2.1). Die Zeilen, in denen sich diese Tags befinden, zählen *nicht* zum Loop selbst.

1.4.1 Identifikation lokaler Loops

Steht in der Zeile eines LOOPSTART Tags nach dem identifizierenden Tag weitere non whitespace Zeichen, werden diese als Loop Name interpretiert. Mittels dieses Namens kann man von anderen Loos oder sogar aus anderen Files diesen Loop referenzieren.

1.4.2 Import aus Datei

Enthält der Loop Name ein @ Charakter, wird der Loop als Referenz interpretiert. Der lokale Inhalt des Loops wird dabei ignoriert und anstelle dessen der Loop-Inhalt aus der referenzierten Loop genommen. Referenzen folgen dem Format:

$$LOOPSTART\ loopname@sourcefile$$

Bsp:

```
#region HYCALPER LOOPSTART initialize@_sum.cs
#endregion HYCALPER LOOPEND initialize@_sum.cs
```

Hier wird auf den Inhalt des Loops mit dem Namen 'initialize' innerhalb der Datei '_sum.cs' referenziert.
Wie im letzten Beispiel kann der Name des Loops oder der Reference in der LOOPEND Zeile wiederholt werden. Dies ist optional, dient aber der Übersichtlichkeit und ist daher zu empfehlen. Die Angabe des Files, aus dem Loop gelesen werden soll, kann entweder als absolute oder als relative Pfadangabe erfolgen. Bei einer relativen Pfadangabe werden die folgenden Pfade in dieser Reihenfolge durchsucht:

- das Verzeichnis des aktuellen Files, welches Hycalper bearbeitet
- das Verzeichnis der Hycalper exe.

Beachte: Referenzen bewirken nur, daß der Loop Inhalt eines fremde Files an dieser einen Stelle (lokaler Loop) verwendet wird. Als Typdefinitionen dient aber immernoch der XML Bereich aus dem lokalen File!

1.4.3 Pattern aus Loop ausnehmen

Enhält der Loop Name eine Sequence wie

$$HCEXCLUDE = 1, 2, 3, 4, 6$$

, so werden duese Idicies aus der Loop Generierung ausgeschlossen. Das EXCLUDEPATTERN ist per Default auf 'HCEXCLUDE' (GroßSchreibung beachten!) eingestellt. Dies kann im HycalperConsole Quelltext geändert werden.

1.5 Enumerations

Enums zählen alle <destination> Tags eines XML types auf. Dies findet vor allem Verwendung in XML-Source-Dokumentationen. Beispielsweise kann man so alle Typen aufzählen, für die eine Funktion überladen ist.
Das ENUMPATTERN ist das Tag zur Identifikation der Positionen innerhalb des Source Kodes, an welchen diese Aufzählungen eingesetzt werden sollen. Es kann bislang nur innerhalb des Hycalper Source Kodes geändert werden. Beim Ersetzen wird als Startpunkt das erste Zeichen des Wortes genommen, in welchem sich das Tag befindet. Als Endpunkt das Ende des Wortes. Ein enum-Wort besteht aus folgenden Bestandteilen:

$$ENUMPATTERN\ key:$$

Dabei gibt der Key an, aus welcher XML Typedefinition alle <destination>-Tags aufgezählt werden sollen. Dieser Key muß lokal im XML abschnitt der Datei definiert worden sein. Im Default ENUM-Wort wird der Key also mit colons eingeschlossen. Bei der Ausgabe der Aufzählung, werden die Tags mit Komma getrennt und nach das ENUM Wort in den Text eingefügt. Am Ende folgt kein Komma. Default ENUMPATTERN: `!HC:ENUM:`.
Bsp.: `ENUM Wort: /*!HC:ENUM:inputType:*/`
Das Verhalten von Enumeration innerhalb eines Templates, welches sich selbst als Ausgabe ersetzt ist nicht definiert!

1.6 Ausgabe ins selbe File

Wird die Angabe eines Outfiles im ersten XML Tag ausgelassen, so werden die modifizierten Loops direkt an deren lokale Definitionen angehängt und der Output ins selbe File gespeichert. Vorteil hierbei ist, daß man direkt das Template in eine Projektstruktur einbinden, kompilieren, debuggen und modifizieren kann. Modifikationen innerhalb des Projektes müssen nicht mehr in das externe Template übertragen werden. Ein Nachteil ist die zerbrechlichere Markierung, Die meisten Hycalper Tags müssen im Source Kode verbleiben. Die Loop Sources enthalten aúch im Resultat noch viele Tags und u.U. auch zusätzliche XML-Definitionen. Alle Hycalper einstellungen, die diese Tags

aus der Ausgabe entfernen, sind dementsprechend außer Kraft gesetzt.
Das Verhalten von Enumeration innerhalb eines Templates, welches sich selbst als Ausgabe ersetzt ist nicht definiert!

1.7 vordefinierte Attribute

In der XML Typdefinition muß für jeden Typ bestimmt werden, wie sein Suchstring gefunden wird, bzw. wie der durch ihn referenzierte Inhalt in der Datei ersetzt wird. Die Definition erfolgt mit Hilfe eines `<locate>` - Attributes im `<source>` - Tag der XML Typdefinition (siehe Bsp. 1.3).

1.7.1 after

Zum Ersetzen wird das Wort genommen, welches unmittelbar nach dem Wort steht, in welchem der Suchstring vorkommt. Es ist darauf zu achten, daß das zu ersetzende Wort von beiden Seiten mit mindestens einem Leerzeichen eingeschlossen wird.

1.7.2 here

Das Wort, in welchem der Suchstring vorkommt, wird direkt ersetzt.

1.7.3 nextline

Die gesamte nächste Zeile nach der Zeile, in welchem der Suchstring vorkommt wird ersetzt.

1.7.4 enregion

Der Bereich beginnend mit der Zeile, in welcher der Suchstring vorkommt bis zu der Zeile, in welchem das nächste Vorkommen von `#endregion HYCALPER` gefunden wurde.

1.8 Spezialzeichen

Kommen in einem XML Tag speziellen Zeichen vor, die entweder den umliegenden SourceCode oder die XML Syntax zerbrechen würden, müssen diese Zeichen wie bei XML üblich in `<![CDATA[ ... spezialzeichen ... ]]>` eingeschlossen werden.

2 Settings

2.1 feste Einstellungen

Die folgenden Einstellungen können nur im SourceKode geändert werden (HycalperConsole.cs).

2.1.1 LOOPSTART

Tag zum Finden der Loopstartpunkte. Als Startpunkt wird dann der Beginn der Zeile genommen. Default: “#region HYCALPER LOOPSTART“

2.1.2 LOOPEND

Tag zum Finden der Loopendtpunkte. Als Endpunkt wird dann das Ende der Zeile genommen, in welcher der Tag gefunden wurde. Default: „#endregion HYCALPER LOOPEND“

2.1.3 TYPELIST

Tag zum Finden der XML Konfiguration. Die erste Zeile des XML Abschnittes (Root - Tag) muß sich am Anfang der Zeile befinden, welche der Zeile mit dem TYPELIST Tag folgt. Per default ist TYPELIST ein Alias für „!HC:TYPELIST:“.

2.1.4 ENUMPATTERN

Tag zum Lokalisieren der Enums innerhalb des Source Codes. Per Default gesetzt auf `!HC:ENUM:`.

2.1.5 include secure

[true]: Wenn true, werden die Aufzählungen (Enums) in `<![CDATA[...]]>` Tags eingeschlossen.

2.1.6 parenthesize

[true]: Automatisch generierte Code Abschnitte werden in regionen eingeschlossen, welche Code Folding unterstützen : #region HYCALPER AUTOGENERATED CODE

2.1.7 comment

[true]: User will be warned for altering the output by including a comment into the output file.

2.1.8 remove comments

[true]: Alle Vorkommen von Kommentaren, welche mit `\*!HC` beginnen, werden *nach* der Bearbeitung aus dem Output entfernt. Dies gilt nur, sofern ein outfile tag angeben wurde, der Output also nicht direkt im Source-Template gespeichert wird.

2.1.9 keep tags

[false]: ALle Vorkommen von HYCALPER Marks verbleiben nach der Bearbeitung im Kode.

3 Tips'n Tricks

- Hycalper eignet sich zur Verwendung als Pre-Build Event. Dabei ist unbedingt zu beachten, daß alle beiden Parameter angegeben werden, da sonst der Build Prozeß locken kann! Auch ist abzuwägen, ob wirklich bei jedem Build ein neuer Hycalper Prozeß notwendig ist, da dieser je nach template Konfiguration einige Zeit in Anspruch nehmen kann. Ggf. ist ein manuelles Starten (z.B. per Batch file) vorzuziehen.
- Bei der Angabe eines Verzeichnisses als Eingabepfad, bearbeitet Hycalper nur die Files, welche jünger sind, als die in ihnen definierten Outfiles. Besitzt ein Template keine Outfile-Angabe im ersten XML Abschnitt(1.6), wird dieses File in jedem Fall bearbeitet.