FlowProtocol - Anwenderhandbuch

Wolfgang Maier

19. November 2023

Inhaltsverzeichnis

1 Beschreibung und Hintergrund

1.1 Grundidee

FlowProtocol ist eine Anwendung, mit der man Aufgaben-, bzw. Prüflisten zu ausgewählten Themengebieten anhand einfacher Kontrollfragen erstellen kann. Der Name leitet sich ab aus Flow für „Flussdiagramm“ und „Protokoll“ und steht für das Grundkonzept der Anwendung, anhand eines verzweigten Entscheidungsbaums ein Protokoll zusammenzustellen. Die Anwendung funktioniert dabei wie folgt: Nach Auswahl einer Vorlage wird eine Reihe von Multiple-Choice-Fragen gestellt, mit denen die relevanten Aspekte eines Themas Schritt für Schritt eingegrenzt werden, und aus denen am Ende eine genau abgestimmte Auflistung von Punkten zusammengestellt wird, die beispielsweise als Protokoll verwendet werden können.

Die Grundidee der Anwendung besteht darin, ein Medium zu bieten, um Spezialwissen ohne großen Aufwand zu erfassen und es dann in sehr einfacher Form für andere bereitzustellen, ohne dass es dafür vermittelt und erworben werden muss. Die Erstellung der Vorlagen kann und soll direkt durch die Menschen erfolgen, die über das jeweilige Wissen verfügen und die Anwendung für alle möglich sein, die dieses Wissen benötigen.

1.2 Anwendungsbereiche

Die Anwendungsbereiche für diese Anwendung sind sehr vielfältig. Die ursprüngliche Intention war die Aufstellung von Prüflisten für wiederkehrende komplexe Tätigkeiten, die in vielen einzelnen Aspekten variieren können, sodass jeweils nur eine bestimmte Auswahl von Prüfpunkten relevant ist. Eine Gesamtprüfliste, bei der immer ein Großteil der Punkte hätte ignoriert werden müssen, wäre sowohl aufwendig, als auch fehlerträchtig, und daher nicht effektiv. *FlowProtocol* kann hier durch sein Fragekonzept die für jeden Einzelfall relevanten Aspekte herausfiltern, und auf diese Weise sogar beliebig stark ins Detail gehen. Gleichzeitig hilft dem Anwender die Beantwortung der Fragen auch, sich gedanklich ausführlich mit seiner Aufgabe auseinanderzusetzen. Durch die Vermeidung offener Fragen wird gleichzeitig das Risiko minimiert, dass Dinge vergessen werden. Das Ergebnis ist ein zu 100% relevantes und vollständiges Protokoll.

Ein besonders positiver Nebeneffekt bei der Nutzung von Vorlagen besteht darin, dass durch einfaches Auswählen von Antworten am Ende ein Ergebnisdokument entsteht, das nicht mehr manuell erstellt werden muss. Das Anwendungsgebiet dehnt sich damit prinzipiell auf alle Arten von Dokumenten aus, die sich aus einer Aufzählung von Textbausteinen zusammensetzen und deren Zusammenstellung über eine Reihe iterierter Fragen erfolgen kann. Dies umfasst beispielsweise Bestandsaufnahmen, Analyseprotokolle, Zusammenfassungen, Aufgabenlisten und vieles andere mehr. Für kleinere Systeme lassen sich sogar mit überschaubarem Aufwand Vorlagen erstellen, die nicht nur das Ergebnis einer Analyse zusammenfassen, sondern ergänzend dazu auch schon Lösungsansätze und mögliche weitere Schritte beschreiben. Man kann so vom Prinzip her ein kleines Expertensystem erstellen, das wertvolles Spezialwissen eines Experten in sehr einfacher Form für andere Mitarbeiter verfügbar macht, ähnlich wie ein Chatbot. Das schont wertvolle Unternehmensressourcen und beugt Verfügbarkeitsengpässen vor.

Eine besonders nützliche Anwendung ist die Erstellung von interaktiven Anleitungen. Hierbei werden ebenfalls Rahmenbedingungen und einzelne Textbausteine im Vorfeld abgefragt, und in Abhängigkeit davon die für den vorliegenden Fall relevanten Schritte der

Anleitung ausgegeben. Die Möglichkeit, kontextbezogene und durch Textersetzungen angepasste Codepassagen in die Ausgabe einzufügen, macht *FlowProtocol* zu einem besonders nützlichen Werkzeug in der Softwareentwicklung, wo derartige wiederkehrende Muster zum Tagesgeschäft gehören. Die Kombination von Anleitung und Codegenerator erhöht nicht nur die Effizient, sondern auch die Qualität.

Auch kleinere personalisierte Umfragen lassen sich über Vorlagen realisieren, wenn man die Teilnehmer zur Rücksendung der Ergebnisse instruiert. Es gibt sogar einen Befehl, der bei der Rangfolgenbestimmung von kleineren Listen unterstützt, indem die darin vorkommenden Elemente paarweise gegenübergestellt werden (siehe Abschnitt 5.1). Für wichtige Rangfolgen wie beispielsweise Projektprioritäten ist es natürlich besser, man hat detaillierte und bewährte Kriterien, die sich objektiv auf alle Elemente der Liste anwenden lassen, und die am Ende über einen Zahlenwert zu einer Rangfolge oder Auswahl führen. Auch solche Bewertungen lassen sich mit *FlowProtocol* sehr leicht umsetzen und für alle verfügbar machen.

1.3 Konfiguration

FlowProtocol ist bewusst einfach gehalten und soll es auch bleiben. Als Web-Anwendung kann der Dienst innerhalb einer Einrichtung zentral zur Verfügung gestellt werden. Die einzige notwendige Konfiguration ist die Angabe eines serverseitig verfügbaren Vorlagenordners, in dem sich die Vorlagen befinden (einstellbar über die Eigenschaft TemplatePath in der Datei appsettings.json). Es werden weder Datenbank noch zusätzliche Dienste benötigt. Die Vorlage-Dateien sind normale Textdateien (in UTF-8-Codierung), die die eigentliche Logik enthalten, und die über die Anwendung ausgeführt werden. Die Syntax der Vorlage-Dateien ist ebenfalls sehr einfach und bewusst für die manuelle Erstellung in einem Editor vorgesehen. Der Aufbau ist zeilenbasiert und verwendet Einrückung zur Abbildung der Verschachtelung. Die Erstellung einer ersten eigenen Vorlage ist innerhalb von 2 Minuten möglich. Eine ausführliche Beschreibung des Sprachumfangs wird in der Befehlsreferenz gegeben. Das Ergebnis einer Bearbeitung wird am Ende auf dem Bildschirm ausgegeben, von wo aus es in die Zwischenablage übernommen werden kann. Tatsächlich wird in den meisten Fällen eine Weiterverarbeitung in einem anderen System (E-Mail, CRM, Projektverwaltung) erfolgen, wo noch Begriffe ergänzt und Prozesse angestoßen werden können und eine revisionssichere Verwaltung möglich ist.

Die Organisation der Vorlagen erfolgt auf zwei Ebenen. Auf unterster Ebene ist eine Unterteilung in Anwendergruppen vorgesehen, die jeweils ihre eigenen Vorlagen verwenden. In einem Unternehmen oder einer Einrichtung wird sich diese Aufteilung normalerweise am Mitarbeiterorganigramm orientieren. Ein Beispiel wäre die Aufteilung in Vertrieb, Marketing, Entwicklung, Administration. Jeder Ordner im Vorlagenordner wird in *FlowProtocol* als Anwendergruppe auf der Seite Anwendergruppen angezeigt. Innerhalb dieser Ordner kann man die Vorlagen entweder direkt ablegen oder in weitere Unterordner unterteilen, um fachliche Gruppen zu bilden. Der Pfad einer Vorlage innerhalb dieser Struktur findet sich auch in der URL wieder, die innerhalb der Anwendung erzeugt wird, und kann so auch direkt als Link in einer E-Mail, einem Vorgang, im Wiki oder an einer anderen Stelle im Intranet bereitgestellt werden. Der Aufruf führt somit sofort zur Ausführung der gewünschten Vorlage, ohne dass diese über das Menü ausgewählt werden muss.

Ordner auf Ebene der Anwendergruppen, die mit einem _-Zeichen beginnen, werden nicht

im Menü angezeigt. Ruft man den analog zu den anderen Ordnern aufgebauten Link auf, kann man den Inhalt des Ordners jedoch ganz normal über die Oberfläche von *FlowProtocol* anzeigen lassen und die Vorlagen darin auch aufrufen. Auf diese Weise kann man zum Beispiel persönliche Vorlagen in einem eigenen Ordner verwalten, ohne dass die Auflistung der Anwendergruppen dadurch an Übersicht verliert. Ein Schutz vor dem Aufruf durch andere Anwender ist dadurch jedoch nicht gegeben.

Eine weitere Aufteilung kann innerhalb einer Vorlage realisiert werden, z.B. indem man dort auf oberster Ebene mit einem Menü oder einer Einstiegsfrage „Was möchten Sie tun?“ beginnt. Für jede Auswahl kann wieder praktisch eine eigene Vorlage ausgeführt werden. Auch hier wird die Menüauswahl in der URL verwaltet und kann über Links oder Lesezeichen übermittelt oder gespeichert werden.

1.4 Technische Ergänzungen

FlowProtocol basiert auf den .NET-Framework 6.01 und kann sowohl unter Windows, als auch Linux ausgeführt werden. Der Umgang mit Dateipfaden und Zeilenumbruchzeichen wurde für beide Systeme kompatibel gehalten. Die Anwendung arbeitet vollständig zustandslos in dem Sinne, dass bei der Benutzung keinerlei Daten durch die Anwendung gespeichert werden. Es gibt weder eine Datenbank, noch eine Benutzerverwaltung und der Zugriff auf das Dateisystem erfolgt nur lesend. Die Verwaltung der gegebenen Antworten erfolgt vollständig in der URL, was die Möglichkeit bietet, zurückzuspringen oder einen Zwischenstand als Lesezeichen zu speichern oder zu versenden, allerdings werden Aufrufe von URLs in Unternehmen teilweise durch die IT-Infrastruktur protokolliert, sodass eine Verarbeitung schützenswerter Daten auf jeden Fall dahingehend betrachtet werden sollte.

FlowProtocol steht unter der MIT Lizenz und ist unter

```
https://github.com/maier-san/FlowProtocol
```

frei verfügbar.

Viel Freude beim Erstellen von Vorlagen und deren Anwendung!

2 Befehlsreferenz

2.1 Grundlagen des Dateiaufbaus

Vorlagendateien können in einem beliebigen Texteditor erstellt werden. Die sehr einfache Sprache verlangt weder Syntax-Hervorhebung, noch Auto-Ergänzung, wobei letzteres bei dem von mir für das Windows-Betriebssystem empfohlenen Editor Notepad++ schon in einem sehr komfortablen Maße gegeben ist. Als Dateiendung für eine Vorlagendatei muss „.qfp“ (Quick Flow Protocol) gewählt werden, damit die Datei über die Anwendung aufrufbar ist.

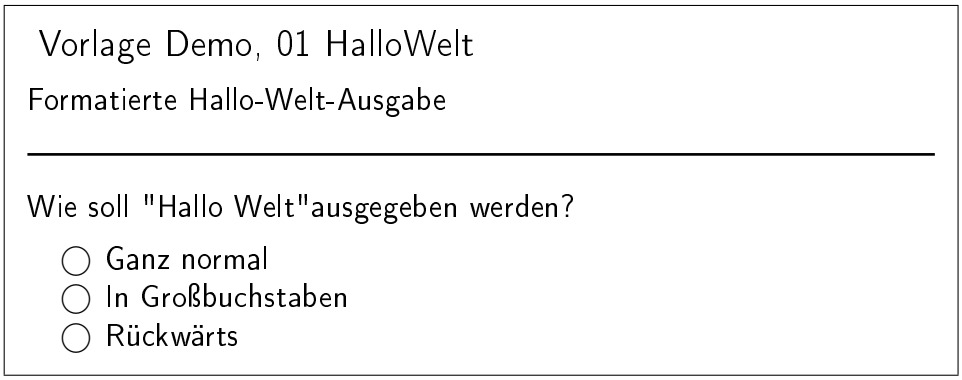
Im Allgemeinen wird die Groß-Klein-Schreibung berücksichtigt, z.B. bei Schlüsseln, Befehlen und Variablen. Bei der Verwendung von Text mit Umlauten sollte darauf geachtet werden, dass diese innerhalb der Anwendung korrekt dargestellt werden. In meiner Umgebung war dies mit der Kodierung UFT-8 der Fall.

Die Verschachtelung der Struktur wird durch Einrückung abgebildet. Hier kann wahlweise mit Tabulator- oder Leerzeichen gearbeitet werden, jedoch sollte dies einheitlich geschehen, da die Struktur bei einer Durchmischung unter Umständen nicht mehr korrekt aufgelöst wird. Die Anwendung ersetzt Tabulatorzeichen durch vier Leerzeichen, unabhängig von ihrer Position. Die Interpretation einer Datei erfolgt zeilenweise, d.h. jeder Zeilenumbruch schließt einen Befehl ab. Der Einsatz von Zeilenumbrüchen zur Formatierung ist damit nicht möglich. Leerzeilen werden bei der Verarbeitung ignoriert und können in beliebiger Menge eingefügt werden.

Beispiel 1

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
    #a3: Rückwärts
        >> tleW ollaH
```

Diese Vorlage führt zu folgender Ausgabe:



Der Anwender kann nun eine der drei Antwortmöglichkeiten wählen, und die Bearbeitung mit der Weiter-Taste fortsetzen, bzw. in diesem Fall abschließen.

Wenn man zum Beispiel die zweite Antwortmöglichkeit „In Großbuchstaben“, so sieht das Ergebnis wie folgt aus:

2.2 Beschreibungstexte

```
/// <Beschreibungstext>
```

Diese Angabe ist nur in der äußersten Ebene einer Vorlagendatei wirksam. Es können mehrere Zeilen dieser Art angegeben werden.

Der angegebene Beschreibungstext wird bei der Darstellung von Fragen im Kopfbereich ausgegeben. Damit lässt sich einem oder mehreren Sätzen kurz beschreiben, wozu die Vorlage dient, zu welchem Zweck sie eingesetzt werden kann.

2.3 Kommentare

Syntax

```
// <Kommentar>
```

Diese Kommentare werden bei der Verarbeitung durch die Anwendung vollständig ignoriert und dienen ausschließlich zur Kommentierung des Vorlagencodes.

2.4 Frage, Antworten und Ausgaben

Syntax

```
?<Frageschlüssel>: <Fragetext>
    #<Schlüssel Antwortmöglichkeit 1>: <Text Antwortmöglichkeit 1>
        >> <Text Ausgabe 1>
    #<Schlüssel Antwortmöglichkeit 2>: <Text Antwortmöglichkeit 2>
        >> <Text Ausgabe 2>
    . . .
```

Das Kernelement der Vorlagen sind Fragen. Diese bestehen aus einem Frageschlüssel und einem Fragetext. Die Schlüssel für Fragen und Antwortmöglichkeiten werden verwendet, um die gegebenen Antworten innerhalb der URL zu verwalten. Sie dürfen nur aus Buchstaben und Zahlen bestehen und sollten möglichst kurz gewählt werden. Die Schlüssel der Fragen müssen über die komplette Vorlage eindeutig sein, die der Antworten nur innerhalb einer Frage.

An der Oberfläche dargestellt wird nur der Fragetext und die Texte der Antwortmöglichkeiten.

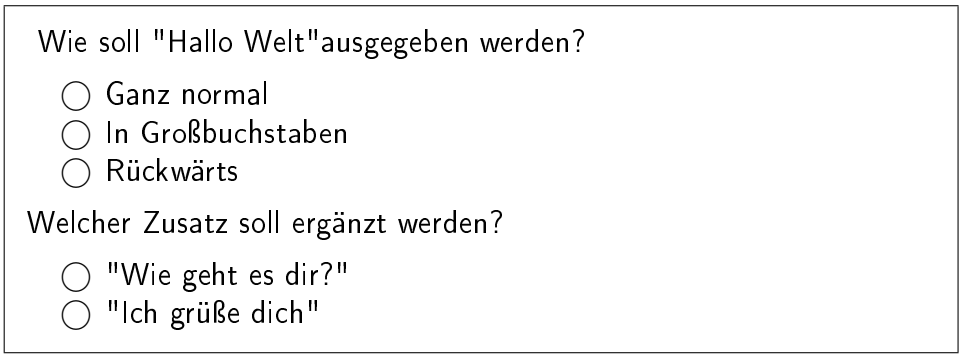
Durch » wird ein Ausgabeeintrag erzeugt, wenn die entsprechende Antwortmöglichkeit gewählt wurde. Die Ausgaben werden als nummerierte Liste zu einem Ergebnis zusammengefasst.

2.5 Sequenzen

Mehrere Fragen können hintereinander aufgelistet werden. Diese werden standardmäßig untereinander auf einer Seite aufgelistet und können zusammen bearbeitet werden.

Beispiel 2

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
    #a3: Rückwärts
        >> tleW ollaH
?F2: Welcher Zusatz soll ergänzt werden?
    #z1: "Wie geht es dir?"
        >> Wie geht es dir?
    #z2: "Ich grüße dich!"
        >> Ich grüße dich!
```



Die Benutzerführung erlaubt es, auch nur einen Teil der angezeigten Fragen zu beantworten und dann die Weiter-Schaltfläche zu betätigen. In diesem Fall werden die nicht beantworteten Fragen einfach auf der Folgeseite erneut angezeigt.

2.6 Verschachtelungen

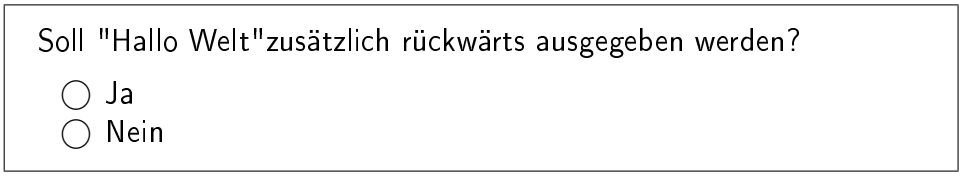
Die Verschachtelung von Fragen ist das Hauptprinzip der Anwendung. Auf diese Weise kann gezielt ins Detail gegangen werden, wenn der entsprechende Fall vorliegt, und man kann Fragen weglassen, wenn sie nicht relevant sind.

Beispiel 3

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
```

```
        >> Hallo Welt
    #a2: In Großbuchstaben
        ?F1a: Text zusätzlich rückwärts ausgeben?
            #j: Ja
                >> TLEW OLLAH
            #n: Nein
                >> HALLO WELT
    #a3: Rückwärts
        >> tleW ollaH
```

Die Darstellung ist zunächst identisch mit der des Ausgangsbeispiels. Die innere Frage wird erst und nur dann angezeigt, wenn in der ersten Ebene die Antwortmöglichkeit „In Großbuchstaben“ gewählt wird.



2.7 Gruppierungen

Das Ergebnis einer Bearbeitung kann in vielen Fällen mehr als eine Liste sein. Eine Vorlage zur Analyse kann zum einen den Ist-Stand erfassen, mögliche Lösungsansätze formulieren und die noch ausstehenden Analyse-Maßnahmen auflisten. Zu diesem Zweck können die Ausgaben in einer sehr einfachen Form gruppiert werden.

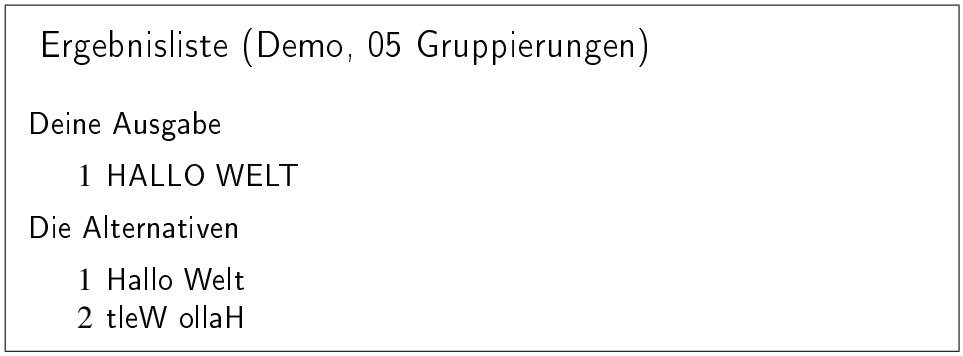
Syntax

```
>> <Gruppe> >> <Text Ausgabe>
```

Beispiel 4

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Deine Ausgabe >> Hallo Welt
        >> Die Alternativen >> HALLO WELT
        >> Die Alternativen >> tleW ollaH
    #a2: In Großbuchstaben
        >> Deine Ausgabe >> HALLO WELT
        >> Die Alternativen >> Hallo Welt
        >> Die Alternativen >> tleW ollaH
    #a3: Rückwärts
        >> Deine Ausgabe >> tleW ollaH
        >> Die Alternativen >> Hallo Welt
        >> Die Alternativen >> HALLO WELT
```

Wählt man die Antwortmöglichkeit „In Großbuchstaben“, so erhält man die folgende Ausgabe:



2.8 Unterpunkte

Ausgaben können je nach Vorlage selbst wieder Tätigkeiten oder Aufgaben beschreiben, die sich aus verschiedenen Schritten zusammensetzen, oder bei denen mehrere Teilaspekte zu berücksichtigen sind. Diese lassen sich in Form von Unterpunkten auflisten. Ein Unterpunkt, der mit der Protokollkennung „https://“ beginnt und als Link erkannt wird, wird dabei in der Ausgabe als ausführbarer Link dargestellt. Man kann sogar ganze Codeblöcke als Unterpunkt formatieren (siehe Abschnitt 2.19).

Syntax

```
>> <Ausgabe Text>
    > <Unterpunkt Text 1>
    > <Unterpunkt Text 2>
    . . .
```

Beispiel 5

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
            > Mehr Infos findest du hier:
            > https://github.com/maier-san/FlowProtocol
    #a3: Rückwärts
        >> tleW ollaH
```

Wählt man die Antwortmöglichkeit „In Großbuchstaben“, so erhält man die folgende

Ausgabe:

Ergebnisliste (Demo, 06 Unterpunkte)

1 HALLO WELT

- Mehr Infos findest du hier:
- https://github.com/maier-san/FlowProtocol

Unterpunkte, die leer sind oder nur Whitespace enthalten, werden komplett weggelassen. Dies kann in Verbindung mit Variablen (siehe Abschnitt 2.15) verwendet werden, um Informationen nur bei Bedarf als Unterpunkte auszugeben.

2.9 Wiederholungen

Eine Frage kann an anderer Stelle wiederholt gestellt werden. Ausschlaggebend dafür ist, dass der Schlüssel der Frage identisch mit einem bereits verwendeten Schlüssel ist. Ist die Antwort auf die Frage bei der Bearbeitung zu diesem Zeitpunkt bereits gegeben, so wird die Fragestellung nicht mehr angezeigt und direkt der jeweilige Antwortzweig verarbeitet. Wenn das aufgrund der Anordnung zwingend der Fall ist, können die Texte abgekürzt werden, da sie ja nicht ausgegeben werden. In diesem Fall empfiehlt es sich zur Verbesserung der Übersichtlichkeit, in den Texten kenntlich zu machen, dass es sich um eine Wiederholung handelt.

Beispiel 6

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
    #a3: Rückwärts
        >> tleW ollaH
?F2: Welcher Zusatz soll ergänzt werden?
    #z1: "Wie geht es dir?"
        ?F1: Wdh. Wie soll "Hallo Welt" ausgegeben werden?
            #a2: In Großbuchstaben
                >> WIE GEHT ES DIR?
            #x: sonst
                >> Wie geht es dir?
    #z2: "Ich grüße dich!"
        >> Ich grüße dich!
```

Bei der Wiederholung einer Frage können auch nur Teile der ursprünglichen Antwortmöglichkeiten gegeben werden. Wird darunter die zuerst gegebene Antwort nicht gefunden, so wird keiner der Antwortzweige verarbeitet. Zur Vereinfachung kann jedoch mit

dem Antwortschlüssel `#x` eine Antwortmöglichkeit gegeben werden, die stellvertretend für alle anderen Antwortmöglichkeiten steht. Wird im obigen Beispiel also für F1 die Antwortmöglichkeit `#a3` gewählt, so wird diese mit der Wiederholung der Frage dem Antwortschlüssel `#x` zugeordnet.

2.10 Implikationen

Die Antwort auf eine Frage kann in manchen Fällen die Antwort auf eine andere implizieren. Dies kann mit dem `~Implies`-Befehl abgebildet werden.

Syntax

```
~Implies <Frageschlüssel 1>=<Antwortschlüssel 1>;  . . .
```

Beispiel 7

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
        ~Implies F2a=j
    #a3: Rückwärts
        >> tleW ollaH
?F2: Welcher Zusatz soll ergänzt werden?
    #z1: "Wie geht es dir?"
        ?F2a: Soll der Zusatz mit Sternchen ausgegeben werden?
            #j: Ja
                >> *** Wie geht es dir? ***
            #n: Nein
                >> Wie geht es dir?
    #z2: "Ich grüße dich!"
        >> Ich grüße dich!
```

Bei Auswahl der Antwort „In Großbuchstaben“ für Frage `F1` wird für die Frage `F2a` die Antwort „Ja“ impliziert. Diese wird dann nicht mehr gestellt, sondern automatisch beantwortet. Im Falle der anderen Antwortmöglichkeiten wird keine Antwort impliziert, sodass die Frage `F2a` in diesem Fall ganz normal gestellt und verarbeitet wird.

2.11 Autonummerierung

Das Durchnummerieren der Frageschlüssel kann in umfangreichen Dateien sehr mühsam werden, da diese zumeist nicht von oben nach unten aufgebaut werden, sondern ausgehend von einer ersten Ebene Stück für Stück in die Tiefe. Die Kontrolle über die Schlüssel benötigt man aber nur, wenn man mit Implikationen oder Wiederholungen arbeitet, oder

die Gültigkeit von Links auch nach Erweiterungen der Vorlage sicherstellen möchte. Für alle anderen Fälle kann man die Schlüssel automatisch vergeben lassen.

Syntax

```
?<Frageschlüsselanfang>': <Fragetext>
```

Beispiel 8

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F': Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
?F': Welcher Zusatz soll ergänzt werden?
    #z1: "Wie geht es dir?"
        >> Wie geht es dir?
    #z2: "Ich grüße dich!"
        >> Ich grüße dich!
```

Die Apostrophzeichen in den Schlüsseln werden pro Datei durchnummeriert und im obigen Beispiel zu `F_1` und `F_2` ersetzt. Schlüssel ohne das `'`-Zeichen bleiben unverändert und können für Implikationen oder Wiederholungen genutzt werden. Implikationen für automatisch nummerierte Schlüssel sind zwar ebenfalls möglich, aber aufgrund der fehlenden Transparenz keine gute Idee.

Bei Verwendung von Funktionen (siehe Abschnitt 2.13) erfolgt die Nummerierung dateiübergreifend und ist unabhängig von der Reihenfolge der Bearbeitung. Dafür wird die Nummerierung im Hintergrund in Form einer Gliederung aufgebaut, die bei tieferen Verschachtelungen eine größere Länge haben. Bei rekursiven Aufrufen mit beliebiger Tiefe wie in Abschnitt 2.14 sollte man daher die Schlüssel lieber explizit mittels Parameter kontrollieren.

2.12 Ausführungen

Bei längeren Sequenzen und insbesondere bei der Verwendung von Wiederholungen und Implikationen ist es wünschenswert, zunächst einen Teil der Fragen vollständig abzuarbeiten, bevor die Fragen aus dem nachfolgenden Teil angezeigt werden. Dies ist mit dem `~Execute`-Befehl möglich.

Syntax

```
~Execute
```

Beispiel 9

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
    #a3: Rückwärts
        >> tleW ollaH
~Execute
?F2: Welcher Zusatz soll ergänzt werden?
    #z1: "Wie geht es dir?"
        >> Wie geht es dir?
    #z2: "Ich grüße dich!"
        >> Ich grüße dich!
```

Die beiden Fragen aus dem Sequenzen-Beispiel werden in diesem Fall einzelnen angezeigt.

2.13 Funktionen

Vorlagen können mitunter sehr umfangreich und tief verschachtelt sein. Verwaltet man solche Vorlagen in einer einzigen Datei, kann das insbesondere aufgrund der Einrückungen sehr unübersichtlich werden, sodass sich Anpassungen und Erweiterungen schwierig gestalten. Hier ist es empfehlenswert, Teile des Codes als Funktionen in jeweils eigene Dateien auszulagern. Eine Funktion ist vom Prinzip und vom Aufbau her das gleiche wie eine Vorlage und wird über eine Datei der Endung „*.qff“ (Quick Flow Function) bereitgestellt. Aufgrund der abweichenden Dateiendung werden Funktionen nicht als Vorlagen in der Anwendung zur Auswahl angeboten.

Die Funktionsdatei muss auf der gleichen Ebene liegen wie die Vorlage, die sie aufruft und der Name darf nur aus Buchstaben und Ziffern bestehen.

Syntax

```
~Include <Name der Funktion>
```

Beispiel 10

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
~Include HalloWelt1
```

Mit der Funktionsdatei HalloWelt1.qff

```
//  Hallo-Welt-Funktion
?F1: Wie soll "Hallo Welt" ausgegeben werden?
```

```
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
    #a3: Rückwärts
        >> tleW ollaH
```

Neben der Erhöhung der Übersichtlichkeit hat die Verwendung von Funktionen viele weitere Vorteile. Auf technischer Ebene vereinfacht es die Verarbeitung, da eine Funktion erst dann geladen wird, wenn der entsprechende Pfad auch tatsächlich erreicht wird. Aus Anwendersicht ergibt sich der große Vorteil, dass man eine Funktion in verschiedenen Antwortzweigen aufrufen kann, und damit keinen doppelten Code pflegen muss. Entsprechend lassen sich Funktionen auch aus verschiedenen Vorlagen heraus aufrufen, sodass fachliche Einheiten ausgelagert und mehrfach wieder verwendet werden können. Mit geteilten Funktionen geht dies sogar für Vorlagen aus verschiedenen Anwendungsgruppen-Ordnern (siehe Abschnitt 2.20).

2.14 Parametrisierte Funktionen

Möchte man eine Funktion innerhalb einer Vorlage mehrfach ausführen, so stößt man auf das Problem, dass die dort eingetragenen Frage-Schlüssel für beide Aufrufe identisch sind, und so der zweite Aufruf automatisch die Antworten des ersten Aufrufs übernehmen würde. Zudem wäre die Anzeige der Fragen identisch, sodass man diese beim Beantworten nicht den jeweiligen Aufrufen zuordnen könnten. Um dieses Problem zu lösen, lassen sich Aufrufe parametrisieren.

Syntax

```
~Include <Name der Funktion> <Parameter 1>=<Wert 1>; . . .
```

Beispiel 11

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
~Include HalloWelt2 weltindex=1; weltbezeichnung=Meine Welt
~Include HalloWelt2 weltindex=2; weltbezeichnung=Deine Welt
```

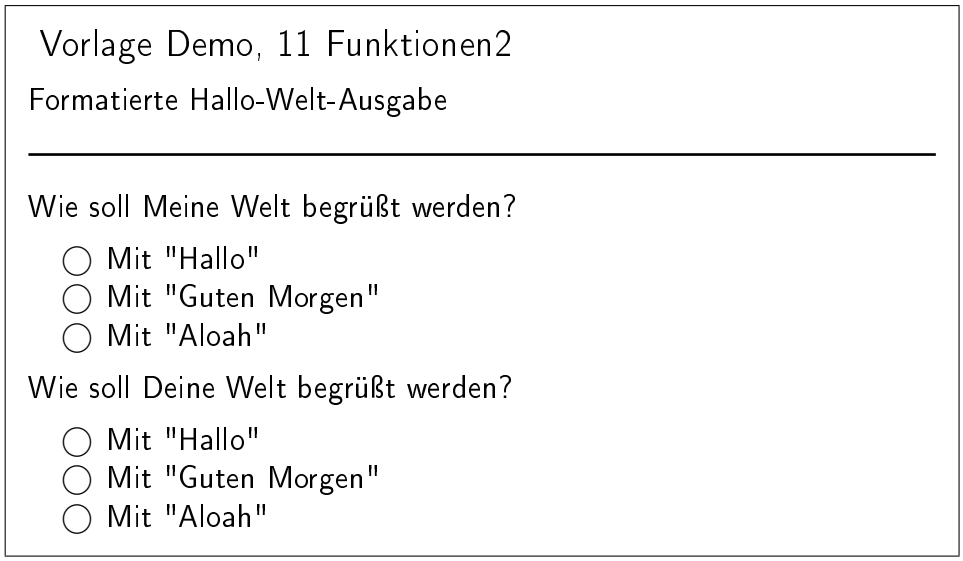
Mit der Funktionsdatei HalloWelt2.qff

```
// Hallo-Welt-Funktion
// Parameter $weltindex: Schlüsselindex
// Parameter $weltbezeichnung: Weltbezeichnung
?F$weltindex1: Wie soll $weltbezeichnung begrüßt werden?
    #a1: Mit "Hallo"
        >> Hallo $weltbezeichnung
    #a2: Mit "Guten Morgen"
        >> Guten Morgen $weltbezeichnung
    #a3: Mit Aloah
        >> Aloah $weltbezeichnung
```

Beim Aufrufen der Funktion HalloWert2 werden in jeder Zeile die mit dem $-Zeichen gekennzeichneten Variablen durch die beim Aufruf zugeordneten Werte ersetzt. Da auch in den Frageschlüsseln die Variable `$weltindex` integriert wurde, ergeben sich bei den beiden aufrufen unterschiedliche Frageschlüssel `F11` und `F21`, über die jeweils eigene Antworten verwaltet werden können. Die Texte der Fragen enthalten ebenfalls Variablen und sind damit klar zuzuordnen.

Es wird empfohlen die Parameter einer Funktion in Form von Kommentaren im Kopfbereich zu beschreiben.

Das obige Beispiel wird wie folgt angezeigt:



Eine eindeutige Benennung der Schlüssel wäre auch automatisch möglich gewesen, jedoch auf Kosten der Möglichkeit, innerhalb einer Funktion wahlweise auch auf Fragen der aufrufenden Stelle zuzugreifen. Deshalb wurde hierauf verzichtet.

2.15 Variablen

Variablen können auch unabhängig von Funktionsparametern gesetzt und in Fragen, Antworten und Ausgaben verwendet werden.

Syntax

```
~Set <Variable 1>=<Wert 1>; <Variable 2>=<Wert 2>; . . .
```

Beispiel 12

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
        ~Set Preis=5 Euro
    #a2: In Großbuchstaben
        >> HALLO WELT
```

```
        ~Set Preis=8 Euro
    #a3: Rückwärts
        >> tleW ollaH
        ~Set Preis=12 Euro
~Execute
?F2: Soll der Preis von $Preis ausgegeben werden?
    #j: Ja
        >> Der Preis beträgt $Preis.
    #n: Nein
```

Wird eine Variable in einer Frage oder Antwort verwendet, sollte durch die Struktur sichergestellt werden, dass die Frage, die für das Setzen der Variablen zuständig ist, zuvor ausgeführt wurde. Dies kann z.B. mit dem oben beschriebenen `~Execute`-Befehl geschehen.

Variablen werden in umgekehrter alphanumerischen Reihenfolge ersetzt, so dass eine Variable `$ab` nicht unbeabsichtgterweise innerhalb der Variablen `$abc` ersetzt wird, wenn bei Variablen gesetzt sind.

2.16 Additionen

Über Variablen lassen sich auch einfache Additionen durchführen, beispielsweise um Kosten, Aufwände oder Komplexität zu ermitteln, die sich aus den gewählten Antwortmöglichkeiten zusammensetzen. Dies ist ebenfalls über den `~Set`-Befehl möglich.

Syntax

```
~Set <Variable 1>+=<Inkrement 1>;  . . .
```

Additionen und Variablenzuweisungen können auch im gleichen `~Set`-Befehl gemischt werden. Bei ihrer ersten Verwendung in einer Addition wird der Variablen zuvor ein Wert von 0 zugewiesen. Hat eine Variable bei der Verwendung in einer Addition bereits einen Wert, der nicht als natürliche Zahl interpretierbar ist, so wird die Addition ignoriert.

Beispiel 13

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
        ~Set Preis=5
    #a2: In Großbuchstaben
        >> HALLO WELT
        ~Set Preis=8
    #a3: Rückwärts
        >> tleW ollaH
        ~Set Preis=12
?F2: Welcher Zusatz soll ergänzt werden?
```

```
    #z1: "Wie geht es dir?"
        >> Wie geht es dir?
        ~Set Preis+=3
    #z2: "Ich grüße dich!"
        >> Ich grüße dich!
        ~Set Preis+=7
?F3: Soll der Preis ausgegeben werden?
    #j: Ja
        >> Der Preis beträgt $Preis Euro.
    #n: Nein
```

2.17 Schleifen

In machen Fällen möchte man eine Funktion wiederholt für eine beliebige Menge an Elementen aufrufen. Dies ist möglich, indem man das Hochzählen von Variablen mit den Parameterzuweisungen eines rekursiven Funktionsaufrufes kombiniert.

Beispiel 14

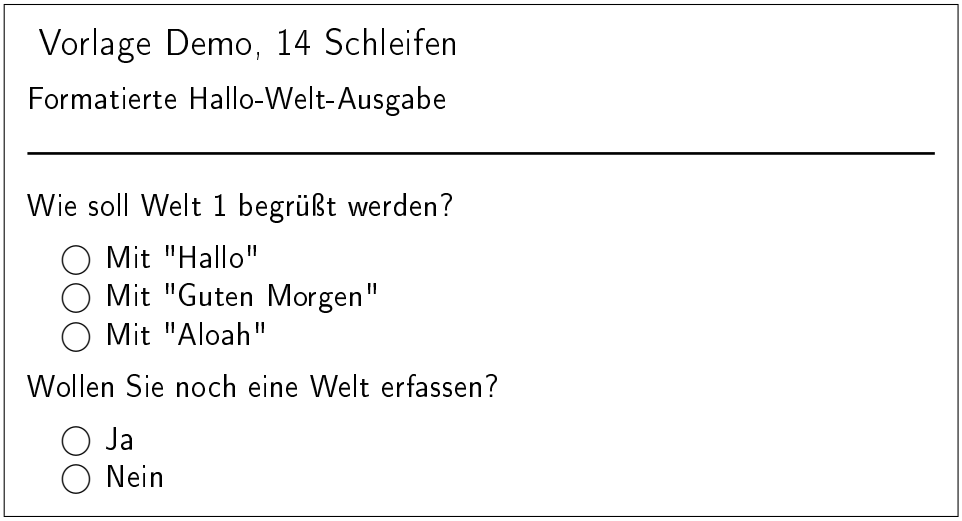
```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
~Include HalloWelt3 weltindex=1
```

Mit der Funktionsdatei HalloWelt3.qff

```
// Hallo-Welt-Funktion
// Parameter $weltindex: Schlüsselindex
?F$weltindex1: Wie soll Welt $weltindex begrüßt werden?
    #a1: Mit "Hallo"
        >> Hallo Welt $weltindex
    #a2: Mit "Guten Morgen"
        >> Guten Morgen Welt $weltindex
    #a3: Mit Aloah
        >> Aloah Welt $weltindex
?F$weltindex2: Wollen Sie noch eine Welt erfassen?
    #j: Ja
        ~Set dummy=$weltindex; dummy+=1
        ~Include HalloWelt3 weltindex=$dummy
    #n: Nein
```

Man beachte, dass Additionen wie im `~Set`-Befehl in den Zuweisungen des `~Include`-Befehls selbst nicht möglich sind, da Include-Parameter anders verarbeitet werden als Variablen.

Das obige Beispiel wird wie folgt angezeigt:



Wählt man nacheinander Antworten und bestätigt die zweite Frage in den ersten beiden Durchläufen mit ja, bekommt man ein Ergebnis wie folgt:

2.18 Eingaben

In manchen Fällen kann es sinnvoll sein, kurze Texte oder einzelne Begriffe abzufragen, um diese als Bestandteil der Ausgabe zu verwenden. Insbesondere wenn man über eine Schleife eine Sequenz von Elementen abfragt, die man in der Ausgabe nicht nur als numerierte, sondern benannte Aufzählung haben möchte. Dafür gibt es den `~Input`-Befehl.

Syntax

```
~Input  <Eingabeschlüssel>: <Fragetext>
```

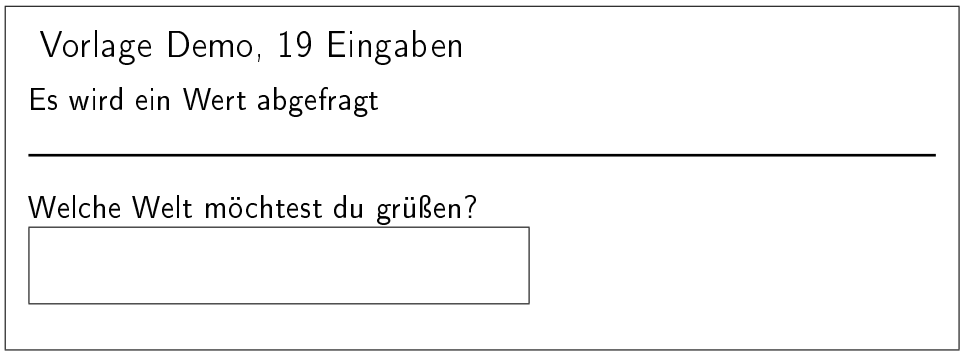
Der Schlüssel dient wie bei den Fragen zur Verwaltung der Eingabe in der URL, und der Fragetext wird an der Oberfläche angezeigt. Der Zugriff auf die Eingabe erfolgt wie bei Variablen über den Eingabeschlüssel, indem man das $-Zeichen voranstellt. Eine Eingabe kann ebenso wie die Antwort auf eine Frage mit dem `~Implies`-Befehl impliziert werden. Die Zeichenzahl ist pro Antwort auf maximal 100 Zeichen beschränkt, und aufgrund der beschränkten Zeichenzahl in der URL sollte diese Funktion nicht überstrapaziert werden.

Beispiel 15

```
/// Es wird ein Wert abgefragt
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Möchtest du eine besondere Welt grüßen?
```

```
    #j: Ja
        ~Input F2: Welche Welt möchtest du grüßen?
    #n: Nein
        ~Implies F2=Welt
~Execute
>> Hallo $F2!
```

Wird im obigen Beispiel die Antwort Ja gewählt, so wird im nächsten Schritt die Frage gestellt, welche Welt man grüßen möchte, mit der Möglichkeit einen Text einzugeben. Wählt man Nein, so wird die Frage implizit mit „Welt“ beantwortet.



2.19 Codebloecke

In der Softwareentwicklung und insbesondere bei der Weiterentwicklung großer Systeme gibt es zahlreiche Komponenten und Bausteine, die immer nach dem grundsätzlich selben Muster auf-, oder eingebaut werden. Oft gibt es dabei Varianten, bei denen man den einen oder anderen Aufruf mehr oder weniger benötigt, und die Benennung der Elemente muss ebenfalls angepasst werden. Es kann die Arbeit deutlich vereinfachen, wenn man mit *FlowProtocol* eine Vorlage für eine Anleitung erstellt, die die relevanten Abhängigkeiten abfragt, und dem Entwickler die notwendigen Implementierungsschritte beschreibt. Noch besser ist es, wenn man eine Stufe weiter geht, und die einzelnen Codefragmente direkt durch die Vorlage erstellen lässt, so dass man diese ausschneiden und in den Produktivcode einfügen kann. Hierfür gibt es die Möglichkeit, ganze Codeblöcke ausgeben zu lassen.

Die Syntax ist ähnlich zu der der Unterpunkte (siehe Abschnitt 2.8). Codeblöcke und Unterpunkte lassen sich auch innerhalb desselben Ausgabeeintrags kombinieren, die Ausgabe erfolgt jedoch blockweise, d.h. zuerst die Unterpunkte und dann der Codeblock.

Syntax

```
>> <Ausgabe Text>
    >| <Codezeile 1>
    >| <Codezeile 2>
    . . .
```

Die Ausgabe erfolgt in einer Festbreitenschriftart und es werden auch führende Leerzeichen mit ausgegeben.

Beispiel 16

```
/// Programmierunterstützung
/// Erstellungsroutine für einen Entitätsdatensatz
//  Anwendungsbeispiel für die Befehlsreferenz
~Input Entity: Wie wird die Entität benannt?
?F1: Von welchem Typ ist der Primärschlüssel?
    #i: Int
        ~Set PTyp=int;
    #G: Guid
        ~Set PTyp=Guid;
~Execute
>> Anleitung >> Implementiere die Funktion Erstelle$Entity()
    >|public $PTyp Erstelle$Entity()
    >|{
    >|   // Rufe generische Funktion auf
    >|   return ErstelleElement<$PTyp>()
    >|}
>> Anleitung >> Ergänze den Aufruf in der Geschäftslogik
    >|   $PTyp id = Erstelle$Entity()
```

Wird im obigen Beispiel als Entitätsbennung „Fahrzeug“ eingegeben und als Typ des Primärschlüssels „int“ gewählt, so bekommt man folgende Ausgabe:

Ergebnisliste (Demo, 21 Codegen)

Anleitung

1 Implementiere die Funktion ErstelleFahrzeug()

```
public int ErstelleFahrzeug()
{
  // Rufe generische Funktion auf
  return ErstelleElement<int>()
}
```

2 Ergänze den Aufruf in der Geschäftslogik

```
int id = ErstelleFahrzeug()
```

2.20 Geteilte Funktionen

Manche Funktionen sind so allgemein, dass man sie anwendergruppenübergreifend verwenden, und deshalb auch zentral verwalten möchte. Zu diesem Zweck kann im Vorlagenverzeichnis ein Ordner mit der Bezeichnung *SharedFunctions* angelegt werden, der beim Aufrufen von Funktionen speziell berücksichtigt wird. Wird die Datei zu einer aufgerufene Funktion im lokalen Ordner nicht gefunden, so wird diese als nächstes im SharedFunctions-Verzeichnis gesucht. Damit lassen sich Funktionen teilen und bei Bedarf auch später noch in das lokale Verzeichnis kopieren und dort anpassen, ohne dass dafür andere Vorlagen angepasst werden müssen.

Im Normalfall wird man geteilte Funktionen nicht so aufbauen, dass sie selber Ausgaben erzeugen, sondern vielmehr so, dass sie ein Ergebnis in einer Menge von Variablen speichern, das dann an der aufrufenden Stelle weiterverarbeitet werden kann.

Das nacholgende Beispiel demonstriert diese Möglichkeit gleich mit zwei Aufrufen derselben Funktion, durch die mittels Parametrisierung zwei unterschiedliche Variablen belegt werden:

Beispiel 17

```
/// Formatierte Hallo-Welt-Ausgabe
//  Anwendungsbeispiel für die Befehlsreferenz
~Include HalloWelt5 weltindex=1; weltbezeichnung=Geteilte Welt
~Include HalloWelt5 weltindex=2; weltbezeichnung=Gemeinsame Welt
>> $Gruss1
>> $Gruss2
```

Aufgerufen wird hierbei die Funktionsdatei HalloWelt5.qff:

```
// Hallo-Welt-Funktion
// Parameter $weltindex: Schlüsselindex
// Parameter $weltbezeichnung: Weltbezeichnung
?F$weltindex1: Wie soll $weltbezeichnung begrüßt werden?
    #a1: Mit "Hallöchen"
        ~Set Gruss$weltindex=Hallöchen $weltbezeichnung
    #a2: Mit "Guten Morgen"
        ~Set Gruss$weltindex=Guten Morgen $weltbezeichnung
    #a3: Mit Aloah
        ~Set Gruss$weltindex=Aloah $weltbezeichnung
```

Ein schöner Anwendungsfall für eine geteilte Funktion ist beispielsweise die Zusammenstellung eines Navigationspfades über eine Auswahlsequenz, z.B. um eine Stelle innerhalb einer Software oder anderen Struktur eindeutig zu benennen. Bei einer solchen Funktion wäre die Notwendigkeit einer zentralen Verwaltung aufgrund von Umfang und ständiger Erweiterung enorm wichtig, und trotzdem kann das Ergebnis über eine einfache Variable transportiert werden.

2.21 URL-Codierung

Durch den Einbau von Links auf Webseiten in die Ausgabe ist es auch möglich, Webanwendungan anzusteuern, die über die URL parametrisiert werden können. Die häufigste Anwendung hierbei werden Suchaufrufe sein, aber es können auch komplexere Formularseiten oder auch der lokale Mail-Client über einen mailto-Link angestoßen werden. Ein weiterer möglicher Anwendungsfall wäre der Aufruf einer anderen *FlowProtocol* - Vorlage mit vollständig oder teilweise vorbelegten Frage- und Eingabeschlüsseln.

Die Hürde, die dabei zu überwinden ist, ist die Codierung der Parameter, so dass diese in der URL übergeben werden können. Sofern die Parameter zur Designzeit feststehen, kann man die Codierung über geeignete Webportale bestimmen und im Vorlagecode hinterlegen, doch bei eingegebenen Parametern ist das nicht möglich. Hierfür gibt es den Befehl `~UrlEncode`.

Syntax

```
~UrlEncode <Var 1>; <Var 2>
```

Dem Befehl kann eine Liste von Variablen übergeben werden, die für die Verwednung in einer URL codiert werden.

Beispiel 18

```
/// Kleine Suchmaschine
//  Anwendungsbeispiel für die Befehlsreferenz
~Input S: Suchbegriff
~Set SBeg=$S
~UrlEncode SBeg
>> Bei Google suchen:
    > https://www.google.com/search?q=$SBeg
>> Bei DuckDuckGo suchen:
    > https://duckduckgo.com/?q=$SBeg&ia=web
```

Im Beispiel wird ein Suchbegriff abgefragt und in die Variable `SBeg` übertragen, da die Codierung nur für Parameter, nicht jedoch für Eingabe- und Frageschlüssel möglich ist. Die Variable wird anschließend in die URLs eingebaut und führt bei Eingabe von `C# int?` zu folgender Ausgabe:

Ergebnisliste (Demo, 22 UrlEncoding)

1 Bei Google suchen:
   - `https://www.google.com/search?q=C%23%20int%3f`

2 Bei DuckDuckGo suchen:
   - `https://duckduckgo.com/?q=C%23%20int%3f&ia=web`

Die häufig in der Softwareentwicklung verwendete Anwendung Jira bietet z.B. eine Webseite, über die Vorgänge mit vorbelegten Eigenschaften erstellt werden können, wobei Projekt, Typ und beliebige Eigenschaften per Parameter vorbelegt werden können. Ein Expertensystem könnte so durch gezielte Fragen die notwendigen Informationen sammeln und am Ende in der Ausgabe die Erstellung eines Vorgangs vom richtige Typ mit passenden Eigenschaften als Link anbieten.

2.22 Links mit Anzeigetext

Aus Abschnitt 2.8 wissen wir, dass Unterpunkte, die als URL interpretiert werden können, ausführbar ausgegeben werden. Die zuletzt gezeigten Möglichkeiten führen dabei jedoch in der Regel zu sehr langen und kryptischen Links, für die ein alternativer Anzeigetext wünscheswert wäre. Dies ist wie folgt möglich:

Syntax

```
>> <Ausgabe Text>
    > <URL 1>|<Displaytext 1>
    > <URL 2>|<Displaytext 2>
    . . .
```

Der URL-Angabe wird, abgetrennt durch das Pipe-Zeichen der anzuzeigende Text nachgestellt. Wird der Zusatz weggelasen, so wird die URL vollständig wie in Abschnitt 2.8 beschrieben ausgegeben. Ist der vordere Teil nicht als URL interpretierbar, so wird der Anzeigetext ignoriert.

Beispiel 19

```
/// Der Textbaustein "Hallo Welt" wird ausgegeben
//  Anwendungsbeispiel für die Befehlsreferenz
>> HALLO WELT
    > Mehr Infos findest du hier:
    > https://github.com/maier-san/FlowProtocol|FP bei GitHub
```

Der in der Vorlage hinterlegte Link wird damit in der Ausgabe wie folgt angezeigt:

Ergebnisliste (Demo, 23 Displaytext)

1 HALLO WELT

- Mehr Infos findest du hier:
- FP bei GitHub

2.23 Verweise auf die eigene URL

Die mit *FlowProtocol* erzeugte Ausgabe wird häufig in andere digitale Dokumente übertragen. In manchen Fällen kann es dabei sinnvoll sein, die eigene URL in die Ausgabe miteinzubauen, z.B. um spätere Anpassungen der Vorlage nachträglich zu berücksichtigen, oder den Ursprung des Dokuments festzuhalten. Es kann auch sinnvoll sein, nur die URL der Vorlage anzugeben ohne die codierten Frage- und Eingabeschlüssel, um direkt weitere Durchführungen anzubieten. Unabhängig davon bietet die URL auch eine einfache Möglichkeit, das Ergebnis einer komplexeren Abfrage über eine Variable weiterzugeben.

Der Zugriff auf die Information erfolgt über die Systemvariablen `$MyResultURL` für die volltändige URL inklusive der Frage- und Eingabeschlüssel und `$MyBaseURL` für die URL der noch nicht ausgefüllten Vorlage.

Beispiel 20

```
/// Der Textbaustein "Hallo Welt" wird ausgegeben
//  Anwendungsbeispiel für die Befehlsreferenz
```

```
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
    #a3: Rückwärts
        >> tleW ollaH
~Set MRU=$MyResultURL
~UrlEncode MRU
>> FlowProtocol-Link:
    > $MyBaseURL|Weitere Ausführung
    > mailto:xyz@abc.de?subject=W-Vote&body=$MRU|Per Mail abstimmen
```

Das Beispiel oben erzeugt als ersten Unterpunkt einen Link, mit dem die URL der Vorlage aufgerufen werden kann, so dass man einen weiteren Durchlauf machen kann. Als zweiter Unterpunkt wir ein mailto-Link erzeugt, bei der der Mail-Inhalt aus der vollständigen URL besteht, so dass der Mail-Empfänger das Ergebnis abrufen kann. Auf diese Weise kann z.B. ein einfacher Rücksendeweg für Abstimmungen angeboten werden.

2.24 Hilfetexte

Manche Fragen und Eingaben können erklärungsbedürftig sein, insbesondere man man ihnen zum ersten Mal oder nach längerer Zeit als Anwender begegnet. Zusätzliche Erläuterungen könnten in die Fragestellung selbst eingebaut werden, würden aber dort den Lesefluss stören. Aus diesem Grund können für für die o.g. Elemente Hilfetexte hinterlegt werden, die in der Darstellung klar abgegrenzt werden. So wie bei den Unterpunkten ist es auch hier möglich, Links mit Anzeigetext zu hinterlegen (siehe Abschnitt 2.22), um auf evtl. weiterführende Informationen auf einer Wikiseite zu verweisen.

Syntax

```
>& <Hilfetext 1>
>& <Hilfetext 2>
. . .
```

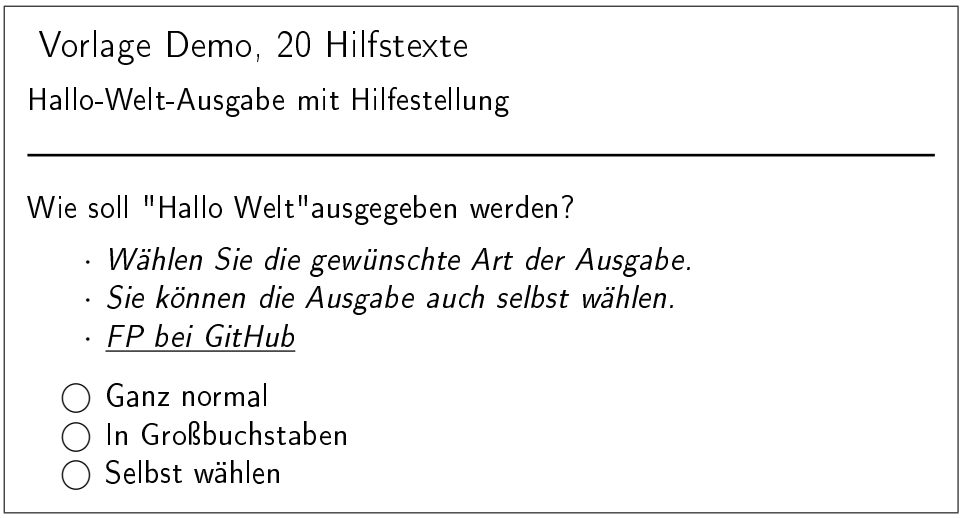
Es können mehrere Hilfetexte hintereinander eingegeben werde. Diese müssen jedoch unmittelbar auf die Frage oder Eingabe folgen und werden nicht über die Einrückung zugeordnet.

Beispiel 21

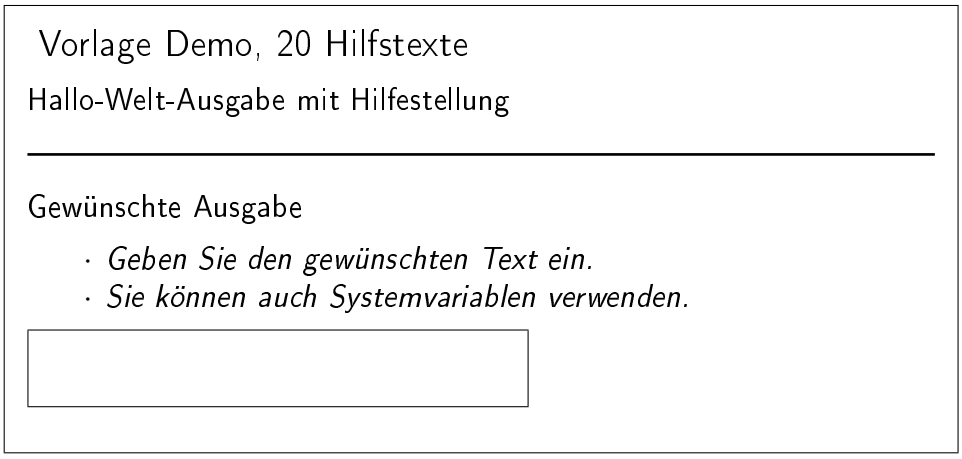
```
/// Der Textbaustein "Hallo Welt" wird ausgegeben
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    >& Wählen Sie die gewünschte Art der Ausgabe.
    >& Sie können die Ausgabe auch selbst wählen
    >& https://github.com/maier-san/FlowProtocol|FP bei GitHub
    #a1: Ganz normal
```

```
    >> Hallo Welt
#a2: In Großbuchstaben
    >> HALLO WELT
#a3: Selbst wählen
    ~Input FO: Gewünschte Ausgabe
        >& Geben Sie den gewünschten Text ein.
        >& Sie können auch Systemvariablen verwenden.
    >> $FO
```

Diese Vorlage führt zu folgender Darstellung:



Bei Wahl der Option „Selbst wählen“ werden die Hilfetexte auch dort bei der Abfrage der gewünschten Ausgabe angezeigt.



2.25 Berechnungen und Rundungen

Neben den einfachen Additionen aus Abschnitt 2.16 kann es manchmal notwendig sein, mit Zahlen und Variablen zu rechnen. Für die Anwendung der Grundrechenoperationen gibt es den Befehl `~Calculate`.

Syntax

```
~Calculate <Variable> = <Wert1> <Operator> <Wert2>
```

Wert 1 und Wert 2 sind Gleitkommaausdrücke in der Formatierung des Systems und der Operator ist eines der Zeichen +, -, *, /. Das Ergebnis wird der angegebenen Variablen zugewiesen.
Zusätzlich gibt es den Befehl ~Round, mit dem ein Wert auf eine bestimmte Anzahl von Dezimalstellen gerundet werden kann.

Syntax

```
~Round <Variable> = <Wert1> | <Genauigkeit>
```

Als Genauigkeit kann eine natürliche Zahl angegeben, was zur Rundund auf entsprechend viele Dezimalstellen werden. Das Ergebnis wird wieder der angegebenen Variablen zugewiesen.

Beispiel 22

```
/// Es wird gerechnet
//  Anwendungsbeispiel für die Befehlsreferenz
~Calculate x = 1,4 + 5,7
~Calculate y = $x * -4,9
~Calculate z = -11 / $y
~Round r = $z|4
>> Ergebnis z = $z
>> Gerundet r = $r
```

Diese Vorlage produziert folgende Ausgabe:

Ergebnisliste (Demo, 26 Berechnungen)

1 Ergebnis z = 0,3161828111526301
2 Gerundet r = 0,3162

2.26 Ersetzungen

Ersetzungen können nützlich sein, um Texteingaben zu verarbeiten oder fest im Code hinterlegte Zeichenkennen wie z.B. Subversion-Eigenschaften in eine andere Form zu bringen. Hierfür gibt es den Befehl ~Replace.

Syntax

```
~Replace <Variable> = <Ausgangstext> |<Suchtext>-><Ersetzungstext>
```

Ausgangstext, Such- und Ersetzungtext sind Zeichenketten und führen dazu, dass alle Vorkommendes Suchtextes im Ausgangstext durch den Erstzungtext ersetz werden. Das Ergebnis wird der angegebenen Variablen zugewiesen.

Beispiel 23

```
/// Es wird ersetzt
//  Anwendungsbeispiel für die Befehlsreferenz
~Set x = Hallo Welt
~Replace y = $x |Hallo->Aloah
~Replace z = $x | ->+
>> Ergebnis y = $y
>> Ergebnis z = $z
```

Diese Vorlage produziert folgende Ausgabe:

Ergebnisliste (Demo, 27 Ersetzungen)

1 Ergebnis y = Aloah Welt
2 Ergebnis z = Hallo+Welt

2.27 Eingabevariablen

Bei Funktionen, die mehrfach in einer Vorlage aufgerufen werden, bietet sich die automatische Nummerierung für Fragen und Eingaben an, um zu vermeiden, diese per Index-Parameter kontrollieren zu müssen (siehe Abschnitt 2.11 und 2.17). Für Fragen ergeben sich dadurch keine Einschränkungen, für Eingaben jedoch schon, da man hier ohne Kenntnis des zur Laufzeit expandierten Eingabeschlüssels nicht auf den Eingabewert zugreifen kann. *FlowProtocol* löst dieses Problem dadurch, dass für Eingabe mit autonummerierten Eingabeschlüssel der Eingabewert zusätzlich in eine Variable übertragen wird, deren Zusammensetzung zur Designzeit bekannt ist.

Syntax

```
~Input  <Eingabeschlüssel>': <Fragetext>
~Set x = $<Eingabeschlüssel>value
```

Es ist klar, dass sich diese Variable beim mehrfachen Aufruf einer Vorlage wiederholt und somit überschrieben wird, aber eine Verarbeitung im lokalen Kontext der Vorlage ist trotzdem möglich. Es empfiehlt sich daher, den festen Teil des Eingabeschlüssels innerhalb einer Funktionsdatei eindeutig zu wählen, und Kollisionen auszuschließen.

Beispiel 24

```
/// Es werden wieder Welten gegrüßt
//  Anwendungsbeispiel für die Befehlsreferenz
?F': Soll die Bergwelt gegrüßt werden?
    #j: Ja
        ~Include HalloWelt5 welt=Bergwelt
    #n: Nein
?F': Soll die Wasserwelt gegrüßt werden?
```

```
    #j: Ja
        ~Include HalloWelt5 welt=Wasserwelt
    #n: Nein
```

mit Funktionsdatei

```
// Hallo-Welt-Funktion
// Parameter $welt
?F': Wie soll die $welt begrüßt werden?
    #a1: Mit Hallo
        >> Hallo $welt
    #a2: Mit Aloah
        >> Aloah $welt
    #a3: Mit eigenem Gruß
        ~Input GF': Grußformel für $welt
        >> $GFvalue $welt
```

Das Beispiel zeigt den typischen Anwendungsfall einer Funktion, die für verschiedene Variablenwerte aufgerufen werden kann, und die auch eine Eingabe pro Aufruf ermöglicht.

2.28 Zufallswerte

Manchmal benötigt man einen Zufallswert, z.B. als Schlüsselwert in Code-Fragmenten oder um den Anwender bei der zufälligen Auswahl einer Stichprobe zu unterstützen. Dieser kann über eine eingebaute Funktion erzeugt werden.

Syntax

```
~Random <Variable> = <Wert1> .. <Wert2>
```

Der Wertebereich besteht aus den ganzzahligen Zahlen im Intervall [`Wert1`,`Wert2`], wobei die Intervallgrenzen eingeschlossen sind.

Beispiel 25

```
/// Die Welt wird nach dem Zufallsprinzip begrüßt
//  Anwendungsbeispiel für die Befehlsreferenz
~Random auswahl=1..3
~Implies F1=a$auswahl
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Hallo Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
    #a3: Rückwärts
        >> tleW ollaH
```

Im Beispiel wir ein zufälliger Wert $x \in \{1,2,3\}$ der Variable `auswahl` zugewiesen und dann, ergänzt durch das Zeichen „a“ der Zielvariablen `F1`. Die Frage wird damit nicht mehr ausgegeben.

2.29 Überschriften

Bei umfangreicheren Vorlagen ist es sinnvoll, die Fragen und Eingaben mit Hilfe von Überschriften zu gliedern.

Syntax

```
++ <Überschrift> ++
```

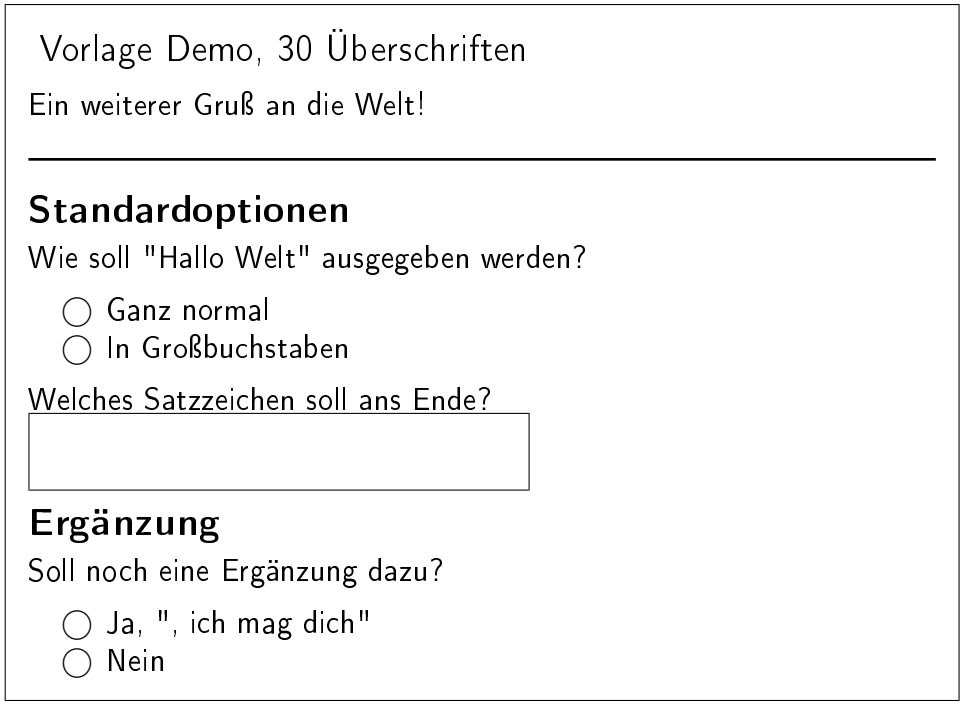
Überschriften können an beliebigen Stellen im Vorlagencode angegeben werden, und werden damit automatisch allen im Vorlagecode nachvolgenden Fragen und Eingaben zugeordnet, bis eine neue Überschrift festgelegt wird. Die Einrückung ist hierbei nicht von Belang.

Bei an der Programmoberfläche wird die Überschrift oberhalb der Frage oder Eingabe ausgegeben, sofern sie sich von der zuletzt ausgegebenen Überschrift unterscheidet. Aufeinanderfolgende Fragen und Eingaben mit der gleichen Überschrift werden also unter derselben Überschrift zusammengefasst.

Beispiel 26

```
/// Ein weiterer Gruß an die Welt!
//  Anwendungsbeispiel für die Befehlsreferenz
++ Standardoptionen ++
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        ~Set gruss = Hallo Welt
    #a2: In Großbuchstaben
        ~Set gruss = HALLO WELT
~Input szeichen: Welches Satzzeichen soll ans Ende?
++ Ergänzung ++
?F2: Soll noch eine Ergänzung dazu?
    #j: Ja, ", ich mag dich"
        ~Set gruss = $gruss, ich mag dich
    #n: Nein
~Execute
>> $gruss$szeichen
```

Dies führt zu folgender Darstellung



Zu beachten ist, dass die Abfolge der Fragen und Eingaben aufgrund der Verschachtelung und der Möglichkeit, nur eine Teilmenge auf einer Seite zu beantworten, von der im Vorlagecode abweichen kann. Die Zuordnung der Überschriften erfolgt unabhängig davon und orientiert sich am Vorlagecode.

2.30 CamelCase-Umwandlung

Möchte man eine Vorlage erstellen, die für eine zu implementierende Programmfunktion die einzubauenden Codepassagen ausgibt, steht man vor der Herausforderung, dass dort Steuerelemente, Klassen oder Methoden in Anlehnung an die Funktion benannt werden sollten, und sich diese Bennungen an Syntaxregeln und Codekonventionen orientieren müssen. Als Programmierer wählt man hier üblicherweise die CamelCase-Schreibweise, wandelt zuvor die Umlaute um und entfernt die Nichtwortzeichen. Diese Arbeit könnte man natürlich auch vom Anwender der Vorlage verlangen und eine entsprechende Eingabe abfragen, da man aber höchstwahrscheinlich schon die sprechende Funktionsbenennung als Eingabewert abgefragt hat, kann man diese auch mit einer eingebauten Funktion in die CamelCase-Schreibweise umwandeln.

Syntax

```
~CamelCase <Variable> = <Wert>
```

Der Wert kann hierbei eine beliebige Zeichenkette sein.

Beispiel 27

```
/// Ein Gruß wird in CamelCase-Schreibweise umgewandelt
//  Anwendungsbeispiel für die Befehlsreferenz
```

```
~Set gruss = Hallo, du süße Honigwelt!
~CamelCase ccgruss = $gruss
>> Vorgabe = $gruss
>> CamelCase = $ccgruss
```

Dies führt zu folgender Ausgabe:

Ergebnisliste (Demo, 31 CamelCase)

1 Vorgabe = Hallo, du süße Honigwelt!
2 CamelCase = HalloDuSuesseHonigwelt

Zu beachten ist, dass innerhalb der Wortzeichen aktuell nur die Umlaute ä, ö, ü, Ä, Ö, Ü und das ß ersetzt werden. Die Ersetzung der diakritischen Zeichen aus dem nichtdeutschen Sprachraum folgt voraussichtlich in einer zukünftigen Version. Bis dahin kann eine solche Ersetzung mit dem `~Replace`-Befehl (siehe Abschnitt 2.26) innerhalb der Vorlage selbst umgesetzt werden.

2.31 Codesequenzen

In Abschnitt 2.19 wird die Möglichkeit beschrieben Codeblöcke in der Ausgabe darzustellen. Zusammen mit der Möglichkeit, dort Variablen zu ersetzen, kann man sehr gut kleinere Programmcodeelemente anpassen und ausgeben. Sobald man jedoch den Programmcode in einem einzigen Block in Abhängigkeit verschachtelter Fragen und Antworten aus kleineren Blöcken zusammensetzen möchte, stößt man hierbei an Grenzen. Hierfür gibt es den Befehl `~InsertCodeSummary`, der eine Sequent aus vorangehenden Codeblöcken in einem Codeblock zusammenfassen kann.

Syntax

```
>> <Ausgabe Text>
    >|~InsertCodeSummary
```

Ergänzend dazu gibt es den Befehl `~StartCodeSummary`, mit dem der Anfang für diese Zusammenfassung gekennzeichnet wird.

Syntax

```
>> <Ausgabe Text>
    >|~StartCodeSummary
```

Beide Befehle wirken nur, wenn sie nicht in Verbindung mit anderen Codezeilen verwendet werden und beziehen sich immer auf die jeweilige Ausgabegruppe (siehe Abschnitt 2.7). Die Befehle selbst werden nicht ausgegeben. Mit Hilfe der beiden Befehle können mehrere Zusammenfassungen innerhalb einer Gruppe erzeugt werden.

Grundsätzlich wird der Code für die Zusammenfassung kopiert, also an der ursprünglichen Stelle und in der Zusammenfassung ausgegeben. Steht der Code jedoch in einer leeren Ausgabe, also einer die selbst keinen Text enthält, so wird die ganze Ausgabe beim Zusammenfassen entfernt, so dass der Code nur noch in der Zusammenfassung erscheint.

Beispiel 28

```
/// Hallo Welt mit Codegenerierung
//  Anwendungsbeispiel für die Befehlsreferenz
>> Codebeispiel >> Kommentar einfügen
    >|// Das kleine Codebeispiel
>> Codebeispiel >>
    >|~StartCodeSummary
>> Codebeispiel >>
    >|public void HalloWelt()
    >|{
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
        >> Codebeispiel >> Ausgabe von Hallo Welt
            >|    Console.Writeline("Hallo Welt");
    #a2: In Großbuchstaben
        >> Codebeispiel >> Ausgabe von HALLO WELT
            >|    Console.Writeline("HALLO WELT");
>> Codebeispiel >>
    >|}
>> Codebeispiel >> Kompletter Code
    > Einfach abtippen:
    >|~InsertCodeSummary
```

In diesem Beispiel beginnt die Zusammenfassung erst in der zweiten Ausgabe, so dass der erste Codeblock nur zusammen mit der ersten Ausgabe ausgegeben wird. Die zweite Ausgabe mit dem Startbefehl ist leer und wird daher nicht ausgegeben. Die dritte Ausgabe ist ebenfalls leer, liegt aber innerhalb der Zusammenfassung und wird durch sie entfernt. Der Prozedurkopf erscheint daher nur in der Zusammenfassung, genauso wie die schließende Klammer am Ende. Der durch die Frage variable Code in der Mitte hängt für beide Antworten an einer nicht leeren Ausgabe und wird somit sowohl dort, als auch in der Zusammenfassung ausgegeben.

Entscheidet man sich für die Großbuchstaben, erhält man folgende Ausgabe:

Ergebnisliste (Demo, 34 Codesequenzen)

Codebeispiel

1 Kommentar einfügen

```
// Das kleine Codebeispiel
```

2 Ausgabe von HALLO WELT

```
    Console.Writeline("HALLO WELT");
```

3 Kompletter Code

- Einfach abtippen:

```
public void HalloWelt()
{
    Console.Writeline("HALLO WELT");
}
```

2.32 Datum- und Uhrzeit-Formatierung

Die Anforderung, in Protokollen auch Datum und Uhrzeit auszugeben, bedarf keiner zusätzlicher Erklärung. Die in Abschnitt 3 beschriebenen Systemvariablen bieten dafür auch schon einige Möglichkeiten. Benötigt man weitere Formate oder möchte man seine Ausgaben unabhängig von den Kultureinstellungen des Servers formatieren, kann man den Befehl `~SetDateTimeFormat` in Verbindung mit der Systemvariablen `$GetFDateTime` verwenden, um beliebige Formate zu erzeugen.

Syntax

```
~SetDateTimeFormat <Formatzeichenfolge>
```

Verwendet werden die Formatzeichenfolgen, die auch in .NET verwendet werden können. Diese werden umfangreich auf zahlreichen Seiten im Internet beschrieben.

Beispiel 29

```
/// Formatiertes Datum
//  Anwendungsbeispiel für die Befehlsreferenz
~SetDateTimeFormat dd. MMM yy
>> Heute ist der $GetFDateTime.
~SetDateTimeFormat dddd
>> Und das ist ein $GetFDateTime.
```

Dieses Beispiel erzeugt die folgende Ausgabe:

Ergebnisliste (Demo, 36 Formatiertes Datum)

1 Heute ist der 28. Jan. 23.
2 Und das ist ein Samstag.

Wie man sieht, kann man die Formatierung zwishen verschiedenen Ausgaben ändern. Um verschiedene Formate innerhalb einer Ausgabe zu verwenden, kann Variablen einsetzen.

Zu beachten ist, dass auch hier die Kultureinstellungen des Servers verwendet werden, was insbesondere bei den Textformaten wie `dddd` usw. einen Unterschied macht. Die beliebige Komposition der Zahlenwerte ist jedoch ohne weiteres möglich.

2.33 Bedingte Zuweisungen

Manche Funktionen oder Berechnungen erfordern die Auswertung von Bedingungen. Mit dem Befehl `~SetIf` kann eine Zuweisung an eine Variable unter einer Bedingung erfolgen, die als logischer Ausdruck angegeben werden kann.

Syntax

```
~SetIf <Variable> = <Wert> <<< <Bedingung>
```

Die Zuweisung erfolgt genau dann, wenn die Bedingung als wahr ausgewertet wird. Im Gegensatz zum ~Set-Befehl kann hier nur eine Variable zugewiesen werden. Die Bedingung wird dabei als Disjunktion von Konjunktionen angegeben, also als Veroderung von Und-Ausdrücken:

```
<Kon1> [|| <Kon2> ...]
```

Jede Konjunktion ist wieder als Und-Verknüpfung von einem oder mehreren Vergleichsausdrücken aufgebaut:

```
<Vgl1> [&& <Vgl2> ...]
```

Ein Vergleichsausdruck kann wiederum folgende Form haben:

```
<s1> == <s2>
<s1> != <s2>
<z1> <> <z2>
<z1> <= <z2>
<z1> >= <z2>
<z1> < <z2>
<z1> > <z2>
<s1> ~ <s2>
<s1> !~ <s2>
```

Hier stehen die Ausdrücke <s1> und <s2> für beliebige Zeichenfolgen und <z1> und <z2> für Werte, die sich zum Auswertungszeitpunkt als Gleitkommazahl interpretieren lassen. Man beachte dabei die Unterscheidung der beiden Ungleich-Operatoren != für Texte und <> für Zahlen. Der Operator ~ steht für „enthält“ im Sinne von, dass die auf der linken Seite stehende Zeichenkennte die auf der rechten Seite als Teil enthält. Entsprechend steht !~ für „enthält nicht“. Klammerungen und unäre Negationen sind nicht Bestandteil der Bedingungssyntax und auch nicht notwendig, da sich über die disjunktive Form und mit den verfügbaren Vergleichsoperatoren alle logischen Kombinationen abbilden lassen. Umschließende Leerzeichen werden vor der Auswertung entfernt und nicht beim Vergleich berücksichtigt. Zu beachten ist auch, dass die Groß-klein-Schreibung von Zeichenketten immer mit berücksichtigt wird, so dass z.B. `A != a` als wahr interpretiert wird.

Anstelle eines Vergleichsausdruckes kann auch direkt ein Wahrheitswert `1` oder `true` für wahr und `0` oder `false` für falsch angegeben werden. Damit können Zwischenergebnisse berechnet und wieder erneut in Bedingungen eingesetzt werden.

Beispiel 30

```
/// Bedingte Zuweisungen
//  Anwendungsbeispiel für die Befehlsreferenz
~Input w: Eine Zahl zwischen 0 und 10 eingeben
~Input t: Ein Wort eingeben
~Execute
~Set z=vom Rand; p=nicht prim;
~SetIf z=aus der Mitte <<< $w>=4 && $w<=6
```

```
~SetIf p=prim <<< $w==2 || $w==3 || $w==5 || $w==7
>> Die Zahl $w ist
    > $z
    > $p
~Set v=kein au oder äu
~SetIf v=au oder äu <<< $t~au || $t~äu
>> Das Wort $t enthält $v.
```

Das obige Beispiel fragt eine Zahl zwischen 0 und 10 und ein Wort ab und bestimmt anhand von drei Bedingungen Eigenschaften, die nachfolgend ausgegeben werden.

Beim Anwenden von bedingten Zuweisungen sollte man darauf achten, dass die Variablen schon zuvor mit dem Wert belegt wurden, der im Fall der nicht erfüllten Bedingung gesetzt sein soll. Versäumt man dies, bleibt die Variable unbelegt und wird in der nachfolgenden Verarbeitung nicht ersetzt.

Die logische Auswertung innerhalb des Ausdrucks erfolgt von links nach rechts und wird jeweils abgebrochen, sobald der Ergebniswert in einer Konjunktion oder der äußeren Disjunktion feststeht.

2.34 If-Bedingungen

Mit Hilfe der bedingten Zuweisungen kann man Bedingungen schon umfangreich nutzen, um Variablen zu setzen. In manchen Fällen möchte man jedoch auch den Verlauf der Ausführung steuern und ähnlich wie bei den Eingaben des Benutzers ganze Abschnitte ausführen oder überspringen. Dies ist mit dem `If`-Befehl möglich:

Syntax

```
~If <Bedingung>
   <Bedingter Code>
```

Die Bedingung wird in der gleichen Weise angegeben wie bei den bedingten Zuweisungen (siehe Abschnitt 2.33) und zum Ausführungszeitpunkt ausgewertet. Als bedingter Code werden alle nachfolgenden Anweisungen interpretiert, die gegenüber dem `If`-Befehl eingerückt sind. Diese werden an die Bedingung gebunden und nur dann ausgeführt, wenn die Bedingungen als wahr ausgewertet wird.

Beispiel 31

```
/// Bedingte Blöcke
//  Anwendungsbeispiel für die Befehlsreferenz
~Input w: Eine Zahl zwischen 0 und 10 eingeben
~Execute
~If $w>=4 && $w<=6
    >> Die Zahl liegt zwischen 4 und 6.
    ?C: Zwischen 4 und 6, toll oder?
        #j: Ja
            >> Toll!
```

```
        #n: Nein
>> Das war es!
```

Das obige Beispiel fragt nach einer Zahl zwischen 0 und 10 und prüft, ob der eingegebene Wert zwischen 4 und 6 liegt. In diesem Fall wird eine entsprechende Ausgabe erzeugt und die Frage „Zwischen 4 und 6, toll oder?“ ausgegeben. Je nach Antwort wird eine weitere Ausgabe erzeugt. Die abschließende Ausgabe „Das war es!“ wird wieder unabhängig von der Bedingung ausgegeben, da sie die gleiche Einrückungstiefe hat, wie die `If`-Bedingung.

Der `Execute`-Befehl vor der `If`-Bedingung ist übrigens notwendig, da die Interpretation der Bedingungen auch hier voraussetzt, dass die dort verwendeten Variablen, in diesem Fall `w`, belegt sind.

3 Systemvariablen

Systemvariablen sind spezielle Variablen, die schon durch das System mit Werten belegt werden. Diese lassen sich wie andere Variablen abrufen und in Fragen, Ausgaben oder Variablenzuweisungen verwenden. Manche der Variablen sind vielmehr als Funktionen zu betrachten und dementsprechend auch mit `Get...` benannt. Im Abschnitt 2.23 haben wir schon zwei Beispiele für Systemvariablen kennengelernt, mit denen die eigene URL abgerufen werden kann.

$NewGuid *Gibt eine zufällige Guid zurück, z.B.* `d817e626-d0b1-40c7-979a-a08b5d6df8a8`.

$GetDate *Gibt das aktuelle Datum in der auf dem System eingestellten Formatierung zurück, z.B.* `29.09.2022`.

$GetDateStamp *Gibt das aktuelle Datum als alphanumerisch sortierbaren Datumstempel zurück, z.B.* `2022-09-22`.

$GetDateTime *Gibt den aktuellen Zeitpunkt mit Datum und Uhrzeit in der auf dem System eingestellten Formatierung zurück, z.B.* `29.09.2022 17:26`.

$GetDayMinutes *Gibt die Anzahl der Minuten des heutigen Tages zurück, z.B.* `721`.

$GetDaySeconds *Gibt die Anzahl der Sekunden des heutigen Tages zurück, z.B.* `43281`.

$GetFDateTime *Gibt den aktuellen Zeitpunkt in der Formatierun zurück, die aktuell über den Befehl* `SetDateTimeFormat` *(siehe Abschnitt 2.32) eingestellt ist.*

$GetTime *Gibt die aktuelle Zeit in der auf dem System eingestellten Formatierung zurück, z.B.* `17:26`.

$GetYear *Gibt das aktuelle Jahr im Format yyyy zurück, z.B.* `2022`.

$MyResultURL *Gibt die URL der aktuellen Vorlageausführung inklusive Schlüsselparameter zurück, siehe auch Abschnitt 2.23.*

$MyBaseURL *Gibt die URL der aktuellen Vorlageausführung ohne Schlüsselparameter zurück, siehe auch Abschnitt 2.23.*

$CRLF *Gibt die Zeichenfolge* `\r\n` *(Carriage Return + Line Feed) zurück, mit der ein Zeilenumbruch auf Windows-System erzeugt wird.*

$LF *Gibt das Zeichen* `\n` *(Line Feed) zurück, mit dem ein Zeilenuburch auf den System Unix/Mac OS X erzeugt wird.*

$ChrXXX *mit* $1 \leq XXX \leq 255$. *Gibt das Zeichen mit dem ASCII-Code XXX zurück, z.B.* `$Chr065 = A`.

$TemplateFilePath *Gibt den Dateipfad der Vorlagendatei zurück.*

$LineNumber *Gibt die Zeilennummer der entsprechenden Codezeile zurück.*

$LineNumber-*n* *mit* $n \in \mathbb{N}$. *Gibt die Zeilennummer der entsprechenden Codezeile abzüglich der Zahl n zurück. Zu beachten ist, dass dieser Audruck keine Leerzeichen enthalten darf.*

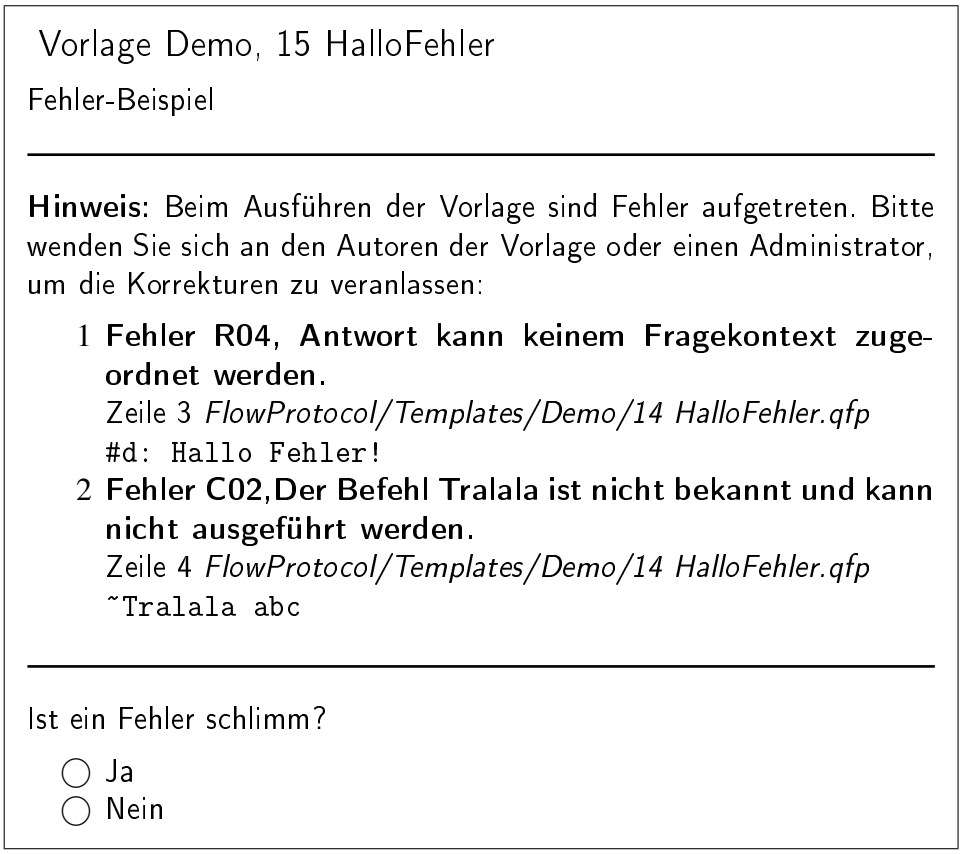
4 Fehlermeldungen

Die Möglichkeit, Vorlagen von Hand zu schreiben birgt ein gewisses Fehlerrisiko. Die Fehler werden bei der Ausführung weitestgehend erkannt und im Kopfbereich mit allen notwendigen Informationen angezeigt.

Beispiel 32

```
/// Fehler-Beispiel
//  Anwendungsbeispiel für die Befehlsreferenz
#d: Hallo Fehler!
~Tralala abc
?F1: Ist ein Fehler schlimm?
    #j: Ja
        >> Stimmt, muss man korrigieren.
    #n: Nein
        >> Stimmt, Ausführung geht weiter.
```

Der Aufruf dieser Vorlage führt direkt zur Anzeige der beiden Fehler:



Wie man erkennt, wird der nach dem Fehler stehende Vorlagencode trotz der Fehler interpretiert und ausgeführt, was die Notwendigkeit einer Korrektur natürlich nicht mindert.

4.1 Einlesefehler

Einlesefehler treten direkt beim Einlesen einer Vorlage- oder Funktionsdatei auf. Da eine Funktionsdatei erst bei Bedarf eingelesen wird, werden auch die darin enthaltenen Fehler

erst angezeigt, wenn der entsprechende Zweig durchlaufen wird.

R01 *Vorlagendatei nicht gefunden.*
Kann auftreten, wenn eine als Lesezeichen gespeicherte URL auf eine nicht (mehr) vorhandene Vorlagendatei verweist, oder kein Lesezugriff auf das Verzeichnis besteht.

R02 *Beschreibungskommentar auf untergeordneter Ebene wird ignoriert.*
Tritt auf, wenn ein Beschreibungskommentar nicht auf oberster Ebene der Vorlage angegeben ist. Dieser wird ignoriert.

R03 *Frage kann keinem Kontext zugeordnet werden.*
Tritt auf, wenn eine Frage keinem Vorlagenkontext zugeordnet werden kann, wenn also die Struktur aus Fragen und Antworten unstimmig ist. Ursache kann eine fehlerhafte Einrückung sein, z.B. weil dabei Leerzeichen und Tabulatorzeichen gemischt wurden.

R04 *Antwort kann keinem Fragekontext zugeordnet werden.*
Tritt auf, wenn eine Antwort keinem Fragekontext zugeordnet werden kann, wenn also die Struktur aus Fragen und Antworten unstimmig ist. Ursache kann eine fehlerhafte Einrückung sein, z.B. weil dabei Leerzeichen und Tabulatorzeichen gemischt wurden.

R05 *Gruppierter Ausgabeeintrag kann keinem Kontext zugeordnet werden.*
Tritt auf, wenn ein gruppierter Ausgabeeintrag keinem Vorlagenkontext zugeordnet werden kann. Mögliche Ursachen siehe Fehler R03.

R06 *Ausgabeeintrag kann keinem Kontext zugeordnet werden.*
Tritt auf, wenn ein nicht gruppierter Ausgabeeintrag keinem Vorlagenkontext zugeordnet werden kann. Mögliche Ursachen siehe Fehler R03.

R07 *Unterpunkt kann keinem Ausgabeeintrag zugeordnet werden.*
Tritt auf, wenn ein Unterpunkt keinem Ausgabeeintrag zugeordnet werden kann. Unterpunkte müssen immer unmittelbar auf Ausgabeeinträge oder andere Unterpunkte oder Codezeilen folgen und dürfen nicht durch Fragen und Antworten unterbrochen werden.

R08 *Execute-Befehl kann keinem Kontext zugeordnet werden.*
Tritt auf, wenn der Execute-Befehl keinem Vorlagenkontext zugeordnet werden kann. Mögliche Ursachen siehe Fehler R03.

R09 *Befehl kann keinem Kontext zugeordnet werden.*
Tritt auf, wenn ein Befehl keinem Vorlagenkontext zugeordnet werden kann. Mögliche Ursachen siehe Fehler R03.

R10 *Zeile nicht interpretierbar*
Tritt auf, wenn eine Zeile nicht als Anweisung interpretiert werden kann, sie also keiner innerhalb einer Vorlage bekannten Syntax entspricht.

R11 *Input-Befehl kann keinem Kontext zugeordnet werden.*
Tritt auf, wenn ein Input-Befehl keinem Vorlagenkontext zugeordnet werden kann. Mögliche Ursachen siehe Fehler R03.

R12 *Codezeile kann keinem Ausgabeeintrag zugeordnet werden.*
Tritt auf, wenn eine Codezeile keinem Ausgabeeintrag zugeordnet werden kann. Codezeilen müssen immer unmittelbar auf Ausgabeeinträge oder Unterpunkte oder andere Codezeilen folgen und dürfen nicht durch Fragen und Antworten unterbrochen werden.

R13 *Hilfetext kann keinem Kontext zugeordnet werden.*
Tritt auf, wenn ein Hilfetext keine Kontext zugeordnet werden kann. Hilfetexte können direkt an eine Frage oder eine Eingabe angefügt werden.

4.2 Ausführungsfehler

Ausführungsfehler treten im Zusammenhang mit der Ausführung von Befehlen auf und sind teilweise vom Ausführungsverlauf abhängig. Auch sie werden erst dann angezeigt, wenn der entsprechende Zweig durchlaufen wird.

C00 *Beim Ausführen eines Befehls ist ein unbehandelter Fehler aufgetreten.*
Tritt auf, wenn bei der Ausführung ein Fehler auftritt, der durch die Anwendung nicht weiter eingegrenzt werden konnte.

C02 *Der Befehl ... ist nicht bekannt und kann nicht ausgeführt werden.*
Tritt auf, wenn eine Befehlsanweisung keinem bekannten Befehl entspricht. Eine Abweichung in der Groß-Klein-Schreibung kann als Ursache ausgeschlossen werden.

C03 *Die Funktionsdatei ... konnte nicht geladen werden.*
Tritt auf, wenn in einem Include-Befehl angegebene Funktionsdatei nicht gefunden werden konnte.

C04 *Der Wert der Variablen ... konnte nicht als ganze Zahl interpretiert werden.*
Tritt auf, wenn für eine Variable im Set-Befehl ein Wert addiert werden soll, und diese zum entsprechenden Zeitpunkt einen nichtnumerischen Wert enthält.

C05 *Der Aufruf der Funktionsdatei ... überschreitet das Rekursionsmaximum von 100.*
Tritt auf, wenn mehr als 100 Mal eine Funktionsdatei aufgerufen wird. Diese einfache Sicherung verhindert Endlosrekursionen, die zum Absturz der Anwendung führen würden.

C06 *Der Ausdruck ... konnte nicht als Berechnungsausdruck interpretiert werden.*
Tritt auf, wenn ein im Zusammenhang mit einen `Calculate`-Befehl angegebener Audruck nicht wie in Abschnitt 2.25 beschrieben interpretiert werden kann.

C07 *Der Ausdruck ... konnte nicht als gültige Zielvariable interpretiert werden.*
Tritt auf, wenn der Audruck links vom Gleichheitszeichen eines `Calculate`-, oder `Round`-Befehls nicht als Variable interpretiert werden kann.

C08 *Der Ausdruck ... konnte nicht als Gleitkommazahl interpretiert werden.*
Tritt auf, wenn einer der Operanden eines `Calculate`-, bzw. der erste Operand des `Round`-Befehls nicht als Gleitkommazahl interpretiert werden kann.

C09 *Division durch 0.*
Tritt auf, wenn über den `Calculate`-Befehl eine Divison durch 0 durchgeführt werden soll.

C10 *Der Ausdruck ... konnte nicht als zulässiger Operator interpretiert werden.*
Tritt auf, wenn der Operator eines `Calculate`-Befehls nicht interpretiert werden kann.

C11 *Der Ausdruck ... konnte nicht als natürliche Zahl interpretiert werden.*
Tritt auf, wenn der zweite Wert des `Round`-Befehls nicht als natürliche Zahl interpretiert werden kann.

C12 *Der Ausdruck ... konnte nicht als Rundungsausdruck interpretiert werden.*
Tritt auf, wenn ein im Zusammenhang mit einen `Round`-Befehl angegebener Audruck nicht wie in Abschnitt 2.25 beschrieben interpretiert werden kann.

C13 *Der Ausdruck ... konnte nicht als Ersetzungsausdruck interpretiert werden.*
Tritt auf, wenn ein im Zusammenhang mit einen `Replace`-Befehl angegebener Audruck nicht wie in Abschnitt 2.26 beschrieben interpretiert werden kann.

C14 *Die angegebenen Werte bilden kein gültiges Intervall.*
Tritt auf, wenn für die beiden für den `Random`-Befehl angegebenen Intervallgrenzen kein gültiges Intervall bilden, also der erste Wert größer ist, als der zweite.

C15 *Die Listendatei ... konnte nicht gefunden werden.*
Tritt auf, wenn beim Fragelisten-Befehl `~ForEach` die angegebene Fragedatei nicht gefunden werden kann. Siehe Abschnitt 5.3.

C16 *Der Ausdruck ... konnte nicht als Vergleichsterm interpretiert werden.*
Tritt auf, wenn bei einer bedingten Zuweisung ein Ausdruck nicht als Vergleichsterm interpretiert werden kann. Siehe Abschnitt 2.33.

C17 *Der If-Befehl konnte nicht vollständig interpretiert werden.*
Beschreibt eine Fehler in Verbindung mit einem `If`-Befehl. Siehe Abschnitt 2.34.

5 Spezialbefehle

Spezialbefehle sind spezielle Befehle, die die Struktur einer Vorlage verändern, und die darauf erfolgten Eingaben in eigener Weise interpretieren und zu einem Ergebnis umsetzen können. Diese Befehle wirken innerhalb des Vorlageteils, wo sie stehen, also auf einer Verschachtelungsebene bis zu den jeweils angrenzenden `~Execute`-Befehlen.

Befehlsübergreifende Fehlermeldungen

S01 *Befehl ohne …-Argument.*
Tritt auf, wenn das zwingende …-Argument nicht angegeben wurde.

S02 *Frageschlüssel … (…-Argument) nicht gefunden.*
Tritt auf, wenn der über das …-Argument angegebene Frageschlüssel nicht gefunden wurde.

S03 *Befehl ohne gültiges …-Argument.*
Tritt auf, wenn für einen Befehl ein ungültiger Parameterwert angegeben wurde, z.B. eine `GroupSize`-Angabe, die nicht als Zahl interpretiert werden kann.

5.1 Rangfolgenbestimmung

Die Rangfolgenbestimmung ist die Implementierung einer Methode, bei der man eine Rangfolge von Dingen wie z.B. Projekten dadurch bestimmt, dass man diese paarweise vergleicht. Bei jedem Vergleich bekommt der Gewinner einen Punkt und am Ende sortiert man die Einträge entsprechend ihrer Punktezahl. Die Reduzierung auf 2er-Vergleiche vereinfacht zwar die Entscheidung pro Frage, macht die Beantwortung jedoch auch deutlich umfangreicher. Für n Elemente ergeben sich $\frac{n(n-1)}{2}$ Einzelfragen, also für $n = 8$ also schon 28 Fragen. *FlowProtocol* unterteilt dabei die Einzelfragen so in Gruppen, dass jedes Element maximal einmal pro Gruppe vorkommt. Für eine gerade Anzahl von n Elementen werden auf diese Weise $n-1$ Gruppen mit $\frac{n}{2}$ Vergleichen gebildet und für eine ungerade Anzahl n sind es n Gruppen mit je $\frac{n-1}{2}$ Vergleichen.

Syntax

```
~Vote Key=<F-Schlüssel> [;GroupName=<G-Name>]
            [;DrawOption=<Unentschiedenoption>]
            [;ResultVar=<Var>; ResultSep=<Trennzeichen>;]
```

Beispiel 33

```
/// Es wird eine Rangfolgenbestimmung durchgeführt
//  Anwendungsbeispiel für die Befehlsreferenz
~Vote Key=V; GroupName=Die besten Welten; DrawOption=egal;
            ResultVar=Erg; ResultSep=$CRLF;
?V: Welche Welt gefällt dir besser?
    #w1: Waldwelt
    #w2: Wasserwelt
```

```
    #w3: Bergwelt
    #w4: Wichtelwelt
~Execute
~UrlEncode ErgRPO
>> Abstimmen >> Klicken Sie auf den Link
    > mailto:xyz@abc.de?subject=W-Vote&body=$ErgRPO|Jetzt abstimmen
```

Der Aufruf dieser Vorlage führt dazu, dass die Frage mit dem Schlüssel `V` zu einer Reihe von Einzelfragen umgestaltet wird. Genauer: Die Frage wird für jede der drei 2er-Kombinationen abgefragt. Die Fragen werden zur 3er-Gruppen zusammengefasst, was hier eine Gruppe ergibt. Das Ergebnis wird unter der Gruppenüberschrift *Die besten Welten* aufgelistet.

Die Option `DrawOption` ermöglicht es, für jede Paarung auch mit unentschieden zu stimmen, wobei der Text für diese Option frei gewählt werden kann. Im Beispiel oben wird als dritte Option jeweils *egal* angezeigt. Die Punktevergabe wird in diesem Fall so angepasst, dass der Sieger einer Paarung 2 Punkte bekommt und im Fall eines Unentschiedens jeder einen Punkt, so dass auch hier die Gesamtpunktzahl immer gleich bleibt.

Mit den Optionen `ResultVar` und `ResultSep` kann festgelegt werden, dass das Ergebnis der Abstimmung in drei Variablen zur Verfügung gestellt wird. Diese beginnen mit der in `ResultVar` angegebenen Zeichenfolge und Enden auf `RPO`, `RPO` und `O`:

Endung	Abkürzung für	Beispiel
`RPO`	Rang, Punkte, Option	Platz 1, 6 Punkte, Bergwelt
`PO`	Punkte, Option	6 Punkte, Bergwelt
`O`	Option	Bergwelt

Im Beispiel oben wird die Variable `ErgRPO` für die Verwendung in einem mailto-Link codiert. Man bechte außerdem den zusätzlichen Strichpunkt nach `$CRLF`. Dieser verhindert, dass die zu einem Whitespace-Zeichen ersetze Zeichensequenz am Zeilenende steht und damit ignoriert wird.

Vorlage Demo, 16 Vote

Es wird eine Abstimmungsbewertung durchgeführt

Welche Welt gefällt dir besser?

- ◯ Waldwelt
- ◯ Wasserwelt
- ◯ egal

Welche Welt gefällt dir besser?

- ◯ Bergwelt
- ◯ Wichtelnwelt
- ◯ egal

Das Ergebnis wird dann wie folgt präsentiert:

Ergebnisliste (Demo, 16 Vote)

Die besten Welten

1 Platz 1 (6 Punkte) Bergwelt
2 Platz 2 (4 Punkte) Waldwelt
3 Platz 3 (2 Punkte) Wasserwelt
4 Platz 4 (0 Punkte) Wichtelwelt

Abstimmen

1 Klicken Sie auf den Link

- Jetz abstimmen

Es kann auch Punktegleichstand geben. Dann teilen sich mehrere Elemente einen Platz.

5.2 Zitatmodus

Sofern es bei einer Vorlage nur darum geht, die gewählten Antworten auf die Fragen zu notieren, kann man sich die Arbeit stark vereinfachen. Mit den standardmäßigen Sprachmitteln müsste man in jedem Antwortzweig Ausgaben erzeugen, die sowohl Frage, als auch Antwort beinhalten, was eine prinzipiell unnötige Wiederholung einer schon vorhandenen Information ist. Der Zitatmodus erledigt genau das für einen kompletten Vorlageteil.

Syntax

```
~Cite [GroupName=<Gruppe>]
```

Beispiel 34

```
/// Es wird gefragt, wie  "Hallo Welt" ausgegeben werden soll
//  Anwendungsbeispiel für die Befehlsreferenz
~Cite GroupName=Antwortprotokoll
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Ganz normal
    #a2: In Großbuchstaben
    #a3: Rückwärts
```

Der Aufruf dieser Vorlage führt dazu, dass die Frage als Ausgabe mit der gewählten Antwort als Unterpunkt ausgegeben wird. Das Ergebnis wird unter der Gruppenüberschrift *Antwortprotokoll* aufgelistet.

Ergebnisliste (Demo, 17 Cite)

Antwortprotokoll

1 Wie soll „Hallo Welt“ausgegeben werden?

- In Großbuchstaben

5.3 Fragelisten

In manchen Fällen besteht eine Vorlage aus einer größeren Menge von Fragen, die sich nur im Text, nicht jedoch in den Antwortmöglichkeiten unterscheiden, und auf die man folglich jeweils dasselbe Frageschema anwenden möchte. Der Fragelisten-Befehl ermöglich eine solche Vorgehensweise und verwendet dabei ähnlich wie die Funktionen eine eigene Datei.

Syntax

```
~ForEach Key=<F-Schlüssel>; List=<Listendatei>; IndexVar=<Var>
            [;Take=<A>] [;GroupBy=<B>] [;GroupFilter=<C>]
            [;ArrayList=<Trennzeichen>]
```

Mit `F-Schlüssel` wird der Schlüssel der Frage angegeben, die für jedes der Elemente in der Liste verwendet werden soll. Mit `Listendatei` wird auf die Datei verwiesen, die die Aufzählung der Elemente enthält. Diese muss sich auf derselben Ebene befinden und als Dateiendung `.qfl` für „Quick Flow List“ haben. `IndexVar` gibt die Variable an, in die die einzelnen Zeilen der Datei eingelesen werden. Diese kann in der gewohnten Notation innerhalb der Frage, den Antworten und den Ausgaben verwendet werden. Durch die optionalen Parameter kann man die Liste auf eine Auswahl von zufällig gewählten `A` Stück beschränken und diese bei der Abfrage in Gruppen von je `B` Stück zusammenfassen lassen. Mit dem Gruppenfilter `C` lässt sich die Auswahl der Elemente aus der Liste auf bestimmte Gruppen beschränken. Diese können mit + aufgezählt werden:

```
GroupFilter=Obst+Gemüde
```

Wird kein Gruppenfilter angegeben, so erfolgt keine Einschränkung durch Gruppen.

Beispiel 35

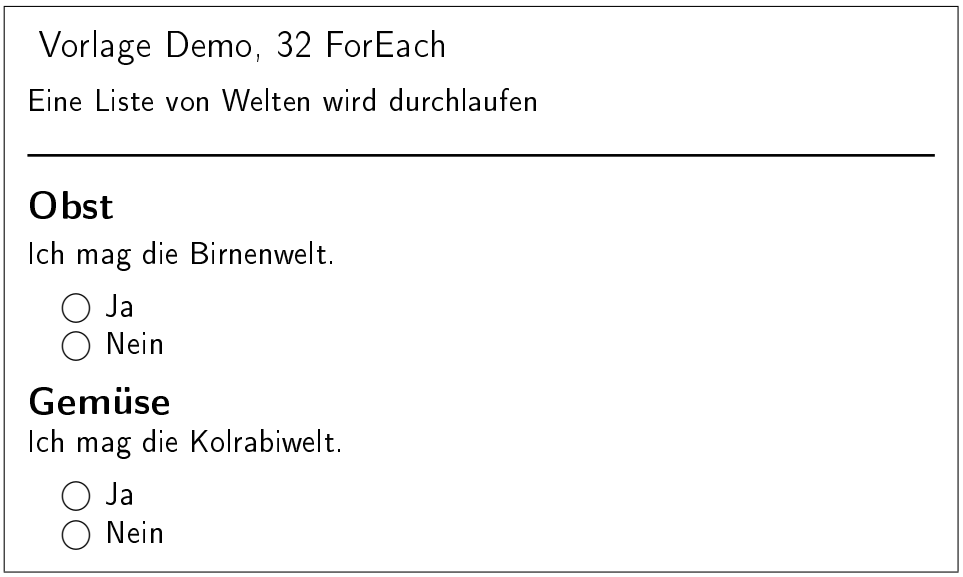
```
/// Eine Liste von Welten wird durchlaufen
//  Anwendungsbeispiel für die Befehlsreferenz
~Set fcount=0; gcont=0
~ForEach Key=F1; List=FEList; IndexVar=idxWelt; Take=4; GroupBy=2;
?F1: Ich mag die $idxWeltwelt.
    #j: Ja
        >> Mag-ich-Welten >> $idxWelt
        ~Set fancount+=1; gcount+=1
    #n: Nein
        >> Sonstige Welten >> $idxWelt
        ~Set gcount+=1
~Execute
>> Auswertung >> Ich mag $fancount von $gcount Welten
```

Das obige Beispiel wählt aus der nachfolgenden Liste 4 Elemente per Zufall aus und gruppiert diese auf zwei Eingabeseiten. Die Liste ist dabei als Datei FEList.qfl mit folgendem Inhalt gegeben:

```
// Aufzählungen für gesunde Welten
[Obst]
Apfel
Birnen
Bananen
Zitronen
[Gemüse]
Blumenkohl
Kolrabi
Zuchini
Kartoffeln
```

Die Syntax der Listendateien erlaubt Kommenarzeilen (`//`) und Gruppierungen mittels eckiger Klammern. Die Gruppierungen werden bei der Eingabe als Überschriften ausgegeben (siehe Abschnitt 2.29) und können dazu verwendet werden, den Inhalt der Liste zusätzlich zu gliedern. Durch die `GroupFilter`-Option kann die Auswahl von Elementen zusätzlich auf einzelne Gruppen beschränkt werden.

Die erste Seite könnte dabei wie folgt aussehen:



Der Parameter `ArrayList` ermöglicht die Angabe eines Trennzeichens, z.B. `|`, mit dem meherere Elemente pro Zeile angegeben werden können. Wird hier ein Zeichen angegeben, so wird jede Zeile der Liste über das Zeichen aufgeteilt und die Teilstücke den Variablen `IndexVar0` bis `IndexVar9` zugewiesen. Es werden maximal diese zehn Variablen zugewiesen. Möchte man den Strichpunkt als Trennzeichen verwenden, so muss man `ArrayList=1` angeben.

Beispiel 36

```
/// Eine Liste von Lebensmitteln
//  Anwendungsbeispiel für die Befehlsreferenz
~Set fcount=0; gcont=0
~ForEach Key=F1; List=ForEachList2; IndexVar=idxL; Take=4;
         GroupBy=2; ArrayList=1
?F1: Mag ich $idxL0?
```

```
#j: Ja, ich mag $idxL0
    >> Probier mal $idxL1.
#n: Nein
```

mit der Liste

```
// Lebensmittel und Gerichte
[Obst]
Äpfel;Apfelkuchen
Birnen;Birnendicksaft
Bananen;Schokobananen
Zitronen;Zitronensorbet
[Gemüse]
Blumenkohl;Blumenkohlpfanne
Kohlrabi;Kohlrabikompott
Zuchini;Zuchiniauflauf
Kartoffeln;Pommes
```

Für die erste Zeile wird die Variable `$idxL0` in der Frage und der Antwortmöglichkeit durch „Äpfel“ ersetzt, und die Variable `$idxL1` in der Ausgabe durch „Apfelkuchen“.

Anwendungsbeispiel

Ein praktischer Anwendungsfall dieser Funktion besteht darin, ein abgegrenztes Wissensgebiet mit Hilfe vieler einzelner Reflektionsaussagen zu beschreiben, und diese dann in Form einer solchen Liste bereitzustellen. Eine solche Reflektionsaussage könnte zum Beispiel lauten „Ich weiß wie man einen neuen Kontextmenüeintrag erstellt.“ Jemand, der feststellen möchte, ob er dieses Wissensgebiet beherrscht, kann sich nun diese Fragen auflisten lassen und den einzelnen Aussagen zustimmen oder nicht. Als Ergebnis bekommt er eine Auswertung über den Grad seines Wissens und eine Auflistung der Aussagen, die seine noch vorhandenen Wissenlücken beschreiben.

Man beachte, dass hier bewusst darauf verzichtet wird, das Wissen explizit durch Vorgabe entsprechender Auswahlmöglichkeiten abzufragen, um die Erfassung der Fragen, die hier ohne jede Syntaxkenntnis möglich ist, so einfach wie möglich zu halten. Durch Einsatz einer zweidimensionalen Liste könnte man jede Aussage um einen Punktewert anreichern oder die Information hinterlegen, von wo oder beim wem das damit verbundene Wissen bezogen werden kann.

Durch die Möglichkeit, sich nur eine kleine zufällige Teilmenge an Aussagen vorlegen zu lassen, hat man zusätzlich auch ein Instrument, um das Wissengebiet regelmäßig stichprobenartig zu überprüfen.

6 Erklärende Beispiele und Anwendungsbeispiele

6.1 Reihenfolge bei Verschachtelungen

Sowohl über die Verschachtelungen von Fragen (siehe Abschnitt 2.6), als auch über den Aufruf von Funktionen (siehe Abschnitt 2.13 und 2.14) werden die Ausgaben in einer verschachtelten Struktur angeordnet, bei der die Aufrufe tieferer Verschachtelungsebenen im Code der Vorlage auch mit einer bestimmten Position, z.B. zwischen Ausgabe *A* und *B* assoziiert sind. Bei der Erzeugung der Ausgabe wird diese Reihenfolge innerhalb der einzelnen Ausgabegruppe wiederhergestellt:

Beispiel 37

```
/// Auf die Reihenfolge kommt es an
//  Anwendungsbeispiel für die Befehlsreferenz
>> Die Begrüßung kann beginnen
?F1: Wie soll "Hallo Welt" ausgegeben werden?
    #a1: Verschachtelt
        >> Hallo Welt
        ~Include Schachtelgruss
        >> Zurück in der alten Welt
    #a2: In Großbuchstaben
        >> HALLO WELT
>> Die Begrüßung ist vorbei
```

mit Funktionsdatei Schachtelgruss.qff

```
>> Start der verschachtelten Begrüßung
?U: Wie soll die verschachtelte Welt begrüßt werden?
    #a1: Mit Hallo
        >> Hallo verschachtelte Welt
    #a2: Mit Aloah
        >> Aloah verschachtelte Welt
>> Ende der verschachtelten Begrüßung
```

Beantwortet man die erste Frage mit „Verschachtelt“ und die zweite mit „Mit Aloah“ ,

dann bekommt man folgende Ausgabe:

Vorlage Demo, 33 Ausgabreihenfolge

Auf die Reihenfolge kommt es an

---

1 Die Begrüßung kann beginnen
2 Hallo Welt
3 Start der verschachtelten Begrüßung
4 Aloah verschachtelte Welt
5 Ende der verschachtelten Begrüßung
6 Zurück in der alten Welt
7 Die Begrüßung ist vorbei

Zu beachten ist, dass sich die Reihenfolge bei gruppierten Ausgaben nur innerhalb der Gruppen auswirkt.

6.2 Selbstlernende Vorlagen

Selbstlernende Systeme begegnen uns viel im Bereich der künstlichen Intelligenz und meinen meist Anwendungen, die sich ihre Fähigkeiten durch wiederholte Versuche selbst beigebracht haben, oder das noch im Laufe ihres Produktivbetriebs tun. *FlowProtocol* ist durch die Möglichkeit, Änderungen und Erweiterungen unmittelbar und sehr einfach in eine Vorlage einzubauen, ebenfalls auf ständige Erweiterungen ausgelegt, wobei auch diese niedrige Schwelle schon für viele zu hoch sein kann. Um es noch einfacher zu machen, eine Vorlage bei Bedarf zu erweitern, kann man die Anleitung für die Erweiterung gleich von Anfang an in die Vorlage einbauen.

Beispiel 38

```
/// Man lernt nie aus
//  Anwendungsbeispiel für die Befehlsreferenz
?F1: Wie soll "Hallo Welt" ausgegeben werden?
   #Normal: Normal
      >> Ausgabe >> Hallo Welt
   #InGrossbuchstaben: In Großbuchstaben
      >> Ausgabe >> HALLO WELT
   // Oberhalb von hier weitere Ergänzungen einfügen
   #ex: Leider nicht in der Liste
      ~Input G1: Grußform
      ~Input G2: Ausgabe
      ~CamelCase CCG = $G1
      >> Ausgabe >> $G2
      >> Erweiterung >> Vorlagedatei öffnen:
         > $TemplateFilePath
         > Code oberhalb von Zeile $LineNumber-8 einfügen:
```

```
>|$Chr009#$CCG: $G1
>|$Chr009$Chr009>> Ausgabe >> $G2
```

Das oben aufgeführte Beispiel bietet zwei Hallo-Welt-Ausgaben an, und ergänzt diese durch einen weiteren Eintrag „Leider nicht in der Liste“. Wählt man diesen aus, werden Grußform und Ausgabe als Eingabe abgefragt, und die Ausgabe in der gewünschten Form erzeugt. Zusätzlich wird eine Anleitung ausgegeben, die genau beschreibt, wie man die Vorlage um diese neue Möglichkeit erweitern kann. Hierbei werden die beiden Systemvariablen `$TemplateFilePath` und `$LineNumber` verwendet, um den Pfad der Datei und die Zeilennummer zum Ausführungszeitpunkt zu bestimmen (siehe Abschnitt 3).

Diese Form der Anleitung bietet sich besonders dann an, wenn die Erweiterung nach einem gut zu beschreibenden Muster verläuft und selbst wieder Artefakte verwendet, die sich gut mit *FlowProtocol* -Vorlagen bereiststellen lassem, wie z.B. der Aufruf von URLs, um Werte nachzuschlagen.

Alternativ zu einer solchen Anleitung kann man auch einen Mailto-Link ausgeben, der die die vermisste Option per Mail an den für die Vorlage zuständigen Mitarbeiter schickt. Dieser kann die Vorlage dann in Ruhe in der gewünschten Form erweitern. Auf diesem Weg läuft man nicht Gefahr, dass es aufgrund von Erweiterungen zu Einschränkungen bei den Vorlagen kommt. Ideal ist es, beide Varianten anzubieten, so dass jeder die Möglichkeit nutzen kann, die er sich selbst zutraut.

6.3 Programmtechnische Möglichkeiten

FlowProtocol ist sicher nicht die erste Wahl und auch nicht dazu gedacht, um komplexe Berechnungen oder Algorithmen zur Datenverarbeitung zu implementieren, aber in manchen Fällen bietet es sich an, und es ist mehr möglich, als man auf den ersten Blick denkt.

Das folgende Beispiel zeigt eine Lösung für eine solche Aufgabenstellung, die man in den üblichen Programmiersprachen sicherlich anders gelöst hätte, und demonstriert, dass man mit den hier verfügbaren Befehlen ebenfalls einiges anstellen kann.

Die Aufgabenstellung besteht darin ein achtstelliges Passwort aus zufälligen Zeichen zu generieren, das mindestens einen und maximal zwei Großbuchstaben, mindestens eine und maximal zwei Ziffern und mindestens ein und maximal zwei Sonderzeichen enthält. Hintergrund dieser Anforderung ist der, dass viele Richtlinien für sichere Passwörter eine Mindestlänge von acht Zeichen und die Verwendung von Groß- und Kleinbuchstaben, sowei Ziffern und Sonderzeichen fordern, und man umgekehrt nicht zuviel davon haben möchte, weil sonst zu viele Zeichen mit der Umschalttaste eingegeben werden müssen.

Um den Lösungsansatz mit einer überschaubaren Menge von Codezeilen zu zeigen, reduzieren wir die Aufgabe auf die Erstellung eines dreistelliges Passworts, das genau einen Großbuchstaben, und mit mindestens 50% Wahrscheinlichkeit auch eine Ziffer enthält.

Beispiel 39

```
/// Erstelle ein sicheres Passwort
//  Anwendungsbeispiel für die Befehlsreferenz

// Ausgangsmaske
```

```
~Set pw=123

// Ersetze die Ziffern von 1 bis 3 zufällig durch ^A bis ^C
~Random pos=1..3
~Replace pw=$pw|$pos->^A
~Replace pw=$pw|3->$pos
~Random pos=1..2
~Replace pw=$pw|$pos->^B
~Replace pw=$pw|2->$pos
~Replace pw=$pw|1->^C

// Wähle Großbuchstaben für ^A
~Random zwert=65..90
~Set zA=$Chr0$zwert
~Replace pw=$pw|^A->$zA

// Wähle Ziffer für ^B mit 50% Wahscheinlichkeit
~Random zB=0..9
~Random zwert=97..122
~SetIf zwert=0$zwert <<< $zwert<100
~Set zX=$Chr$zwert
~Random ausw=1..2
~SetIf zB=$zX <<< $ausw==1
~Replace pw=$pw|^B->$zB

// Wähle Kleinbuchstaben für ^C
~Random zwert=97..122
~SetIf zwert=0$zwert <<< $zwert<100
~Set zC=$Chr$zwert
~Replace pw=$pw|^C->$zC

// Ausgabe
>> Passwort: $pw
```

Ausganssituation ist eine Ausgangsmaske mit den Ziffern 1 bis 3. Im ersten Schritt generieren wir daraus eine zufällige Permutation der Buchsaben A, B, C, bzw. der Zeichenkombinationen mit vorangestelltem `^`, um später keine ungewollte Wechselwirkung mit den Großbuchstaben des Passworts zu haben. Die Permuation erzeugen wir dadurch, dass wir eine der Ziffern per Zufallsgenerator wählen und diese dann ersetzen. Anschließend ersetzen wir die höchste Ziffer durch die gewählte, damit wieder ein zusammenhängender Zahlenbereich entsteht.

Nun ersetzen wir `^A` durch einen Großbuchstaben, den wir zufällig im Zeichenbereich 65 bis 90 wählen. Anschließend wählen wir eine Ziffer und als Alternative einen Kleinbuchstaben, mit dem wir in Abhängigkeit eines Zufallswertes zwischen 1 und 2 die Ziffer wieder überschreiben und damit am Ende `^B` ersetzen. Danach ersetzen wir `^C` mit einem weiteren Kleinbuchstaben. Beim Erzeugen eines Kleinbuchstabens muss man beachten, dass man für den `$Chr`-Befehl die Ziffer 0 voranstellt, wenn der Zeichencode zweistellig ist.

Die gezeigte Vorlage ist nicht interaktiv, da sie weder eine Eingabe, noch eine Frage enthält. Eine mögliche Erweiterung wäre die übergeordnete Auswahl der Komplexität oder die individuelle Bestimmung, ob und in welcher Vielfachheit bestimmte Zeichengruppen vertreten sein sollen.

6.4 Terminologiesystem

Terminologie ist eine komplexe Angelegenheit, und die Einigung auf bestimmte Benennungen kann ebenso aufreibend sein, die der Weg zur Einhaltung. Aus diesem Grund kosten richtige Terminologiersystemne mindestens fünfstellig.

Das nachfolgende Beispiel zeigt, wie man auch mit *FlowProtocol* ein einfaches, aber dennoch brauchbares Terminologiesystem aufbauen kann:

Beispiel 40

```
/// Terminologie-Checker
//  Anwendungsbeispiel für die Befehlsreferenz
~Input txinp: Text
>> Ergebnis >> Eingegebener Text
    > $txinp
~Execute
// Bestätigungen
~Set txcon=$txinp
~Replace txcon=$txcon|Auswahlfeld->Auswahlfeld(*)
~Replace txcon=$txcon|Kennwort->Kennwort(*)
~Replace txcon=$txcon|Kursstatus->Kursstatus(*)
~Replace txcon=$txcon|Veranstaltungsstatus->Veranstaltungsstatus(*)
// Korrekturen
~Set txout=$txinp
~Replace txout=$txout|die Checkbox->das Auswahlfeld
~Replace txout=$txout|Checkbox->Auswahlfeld
~Replace txout=$txout|Kurs-Status->Kursstatus
~Replace txout=$txout|Passwort->Kennwort
// Warnungen
~Set txwrn=$txinp
// Kontextabhängige Ersetzungen
~If $txout~Status
    ?W: Der Begriff "Status" muss differenziert werden.
        #v1: Kursstatus (Status eines Kurses)
            ~Replace txout=$txout|Status->Kursstatus
        #v2: Veranstaltungssstatus (Status einer Veranstaltung)
            ~Replace txout=$txout|Status->Veranstaltungsstatus
        #w: "Status" mit (!) kennzeichnen
            ~Replace txwrn=$txwrn|Status->Status(!)
            >> Anmerkungen >> "Status" muss differnziert werden.
        #x: "Status" unverändert lassen
// Auswertung
~If $txinp==$txout
```

```
    >> Ergebnis >> Keine Terminologie-Ersetzungen.
~If $txinp!=$txout
    >> Ergebnis >> Terminologie-Ersetzungen:
        > $txinp
        > $txout
~If $txinp!=$txcon
    >> Ergebnis >> Korrekte Benennungen (*):
        > $txcon
~If $txinp!=$txwrn
    >> Ergebnis >> Warnungen (!):
        > $txwrn
```

Eingegeben wird ein Text, der terminologisch geprüft werden soll. In einem ersten Schritt werden dabei die Begriffe mit einen Sternchen gekennzeichnet, die Teil der definierten Terminologie sind. In einem zweiten Schritt werden Benennungen ersetzt, die nicht Teil der definierten Terminologie sind, z.B. die Benennung „Passwort“ durch „Kennwort“. In einem dritten Schritt werden Begriffe behandelt, deren Klärung kontextbezogen erfolgen muss, und die ggf. weitere Abfragen notwednig machen. Im Beispiel oben wird der Begriff „Status“ bemängelt, da dieser in der definierten Terminologie stets weiter differenziert werden sollte. Es werden verschiedene Status zur Auswahl angeboten und auch die Möglichkeit, den Begriff zu kennzeichnen. Am Ende wird ausgewertet und ausgegeben, ob Terminologie-Ersetzungen durchgeführt wurden. Zusätzlich werden die Begriffe gekennzeichnet, die Teil der definierten Terminologie sind, so dass man erkennen kann, ob ein Begriff evtl. noch unbekannt ist und eingepflegt werden sollte.

6.5 Statusbestimmung

Komplexe Objekte wie Projekte oder Kurse können einen Status haben, der vom Vorhandensein bestimmter Artefakte oder der erfolgreichen Abarbeitung bestimmter Aufgaben abhängig ist, und damit auch wieder einen Rückschluss auf diese Gegebenheiten ermöglicht. Meist bauen die Status aufeinander auf, so dass bei erreichen eines Status direkt geprüft werden kann, ob evtl. auch schon der Folgestatus erreicht ist.

Das nachfolgende Beispiel demonstriert, wie man einen von drei Status für ein hier nicht weiter spezifiziertes Objekt über eine Vorlage bestimmen kann, wenn jeder Status gegenüber seinem Vorgängerstatus durch zwei notwendige Kriterien bestimmt wird. Zusätzlich wird ein Status 0 als Ausgangsstatus angenommen, der dann vorliegt, wenn selbst die Kriterien von Status 1 nicht erfüllt sind.

Das besondere an der Vorlage ist, dass sie davon ausgeht, dass bereits ein Status vorliegt, und die Bestimmung des neuen Status genau dort ansetzen soll, anstatt immer wieder bei Null anzufangen und die Kriterien aller Status der Reihe nach abzufragen. In diesem Fall wird sogar davon ausgegangen, dass auch ein Rückschritt auf den Vorgängerstatus möglich ist, wenn die für den aktuellen Status vorausgesetzen Kriterien aktuell nicht mehr alle erfüllt sind. Ein solcher Rückschritt ist gerade bei Projektstatus nicht unüblich.

Der Aufbau der Vorlage besteht aus der nachfolgenden Vorlagedatei

Beispiel 41

```
/// Einer von 3 Status wird anhand von Kriterien zugeordnet
```

```
//  Anwendungsbeispiel für die Befehlsreferenz
~Set S1OK=?; S2OK=?; S3OK=?
?S': Welcher Status soll geprüft werden?
    #S1: Status 1
        ~Include Statuskriterien CC=Status1
    #S2: Status 2
        ~Include Statuskriterien CC=Status2
    #S3: Status 3
        ~Include Statuskriterien CC=Status3
~Execute
~If $Erg==Status0
    >> Ergebnis ist Status 0
    >> Für Status 1 fehlt
        > $S1KritA
        > $S1KritB
~If $Erg==Status1
    >> Ergebnis ist Status 1
    >> Für Status 2 fehlt
        > $S2KritA
        > $S2KritB
~If $Erg==Status2
    >> Ergebnis ist Status 2
    >> Für Status 3 fehlt
        > $S3KritA
        > $S3KritB
~If $Erg==Status3
    >> Ergebnis ist Status 3
```

und der Funktionsdatei Statuskriterien.qff:

```
// Statusbestimmung
// $CC: Eingabestatus
~If $CC==Status1
    ~Set S1OK=j
    ?SB$CC': Kriterium A für Status 1
        #e: erfüllt
            ~Set S1KritA=
        #n: nicht erfüllt
            ~Set S1OK=n
            ~Set S1KritA=Kriterium A1
    ?SB$CC': Kriterium B für Status 1
        #e: erfüllt
            ~Set S1KritB=
        #n: nicht erfüllt
            ~Set S1OK=n
            ~Set S1KritB=Kriterium B1
    ~Execute
    ~If $S1OK==j && $S2OK==?
        ~Include Statuskriterien CC=Status2
```

```
    ~If $S1OK==j && $S2OK==n
        ~Set Erg=Status1
    ~If $S1OK==n
        ~Set Erg=Status0
~If $CC==Status2
    ~Set S2OK=j
    ?SB$CC': Kriterium A für Status 2
        #e: erfüllt
            ~Set S2KritA=
        #n: nicht erfüllt
            ~Set S2OK=n
            ~Set S2KritA=Kriterium A2
    ?SB$CC': Kriterium B für Status 2
        #e: erfüllt
            ~Set S2KritB=
        #n: nicht erfüllt
            ~Set S2OK=n
            ~Set S2KritB=Kriterium B2
    ~Execute
    ~If $S2OK==j && $S3OK==?
        ~Include Statuskriterien CC=Status3
    ~If $S2OK==j && $S3OK==n
        ~Set Erg=Status2
    ~If $S2OK==n && $S1OK==?
        ~Include Statuskriterien CC=Status1
~If $CC==Status3
    ~Set S3OK=j
    ?SB$CC': Kriterium A für Status 3
        #e: erfüllt
            ~Set S3KritA=
        #n: nicht erfüllt
            ~Set S3OK=n
            ~Set S3KritA=Kriterium A3
    ?SB$CC': Kriterium B für Status 3
        #e: erfüllt
            ~Set S3KritB=
        #n: nicht erfüllt
            ~Set S3OK=n
            ~Set S3KritB=Kriterium B3
    ~Execute
    ~If $S3OK==j
        ~Set Erg=Status3
    ~If $S3OK==n && $S2OK==j
        ~Set Erg=Status2
    ~If $S3OK==n && $S2OK==?
        ~Include Statuskriterien CC=Status2
```

Die Vorlagedatei fragt den Ausgangsstatus ab und ruf anschließend die Funktionsdatei mit dem entsprechenden Status als Parameter auf. Dabei werden die Kriterien für diesen

Status abgefragt und geprüft, ob diese vollständig erfüllt sind. Ist dies der Fall, so erfolgt ein rekursiver Aufruf für den Folgestatus, falls nicht, für den Vorgängerstatus. Die Rekursion endet, sobald alle Kriterien für einen Status erfüllt, die für den Folgestatus jedoch nicht erfüllt sind, bzw. wenn die Kriterien für den ersten Status nicht erfüllt, oder die für den letzten Status erfüllt sind. In der Ausgabe, die wieder von der Vorlagedatei übernommen wird, wird dann auch zusätzlich zu dem ermittelten Status ausgegeben, welche der Kriterien für den Folgestatus noch nicht erfüllt sind.

In einem realen Anwendungsfall hat man vermutlich mehr Status und mehr Kriterien mit komplexeren Abhängigkeiten. Denkbar sind auch Konstellationen bei denen einzelne Status in Abhängigkeit von Kriterien übersprungen werden können. Durch die explizite Abbildung der Bedingungen und Sprungziele lässt sich hier so gut wie alles umsetzen.

6.6 Kombinationsbildung

Qualitativ gute Softwareentwicklung setzt an vielen Stellen den Überblick über die verschiedenen Variationen voraus, die im Zuge der Bearbeitung auftreten können. Dies gilt durchgängig von der Entwurfsphase bis hin zum Integrationstest. Insbesondere dann, wenn verschiedene Variationen voneinander abhängig sind und daher prinzipiell alle Kombinationen einzeln betrachtet werden müssen, tut man gut daran, alle kombinatorisch möglichen Fälle systematisch aufzulisten.

Die Abfrage einer solchen Konstellation und die anschließende kombinatoriche Auflistung kann mit der nachfolgenden Vorlage umgesetzt werden:

Beispiel 42

```
/// Erzeugt alle Kombinationen aus mehreren Variationen
//  Anwendungsbeispiel für die Befehlsreferenz
~Include VInput varidx=1
~Execute
~Include VWOutput varidx=1; validx=1; kbw=
```

Zusätzlich benötigt werden Funktionen VInput.qff:

```
// Funktion Variation Input
// Parameter: varidx: Nummer der Variation
~Input Var$varidx: Gib die Variation $varidx an:
~Execute
~Include VWInput varidx=$varidx; validx=1; vtxt=$Var$varidx
~Execute
?NV$varidx: Weitere Variation erfassen?
    #j: Ja
        ~Set dummy=$varidx; dummy+=1
        ~Include VInput varidx=$dummy
    #n: Nein
```

VWInput.qff:

```
// Funktion Variationswert Input
// Parameter: varidx: Nummer der Variation
//            validx: Nummer des Variantionswertes
//            vtxt: Text, der die Variation beschreibt
~Input Var$varidx: Gib die Variation $varidx an:
~Input Val$varidxX$validx: Wert $validx für Variation "$vtxt":
>> Variation $varidx: $vtxt >> $Val$varidxX$validx
?NW$varidxX$validx: Weiteren Wert für "$vtxt" erfassen?
    #j: Ja
        ~Set dummy=$validx; dummy+=1
        ~Include VWInput varidx=$varidx; validx=$dummy; vtxt=$vtxt
    #n: Nein
```

VWOutput.qff:

```
// Funktion Variationswert Output
// Parameter: varidx: Nummer der Variation
//            validx: Nummer des Variantionswertes
//            kbw: Teilergebnis einer Kombination
~If $kbw==
    ~Set reskbw=$Var$varidx: $Val$varidxX$validx
~If $kbw!=
    ~Set reskbw=$kbw / $Var$varidx: $Val$varidxX$validx
?NV$varidx: Weitere Variation erfassen?
    #j: Ja
        ~Set dummy=$varidx; dummy+=1
        ~Include VWOutput varidx=$dummy; validx=1; kbw=$reskbw
    #n: Nein
        >> Kombinationen >> $reskbw
?NW$varidxX$validx: Weiteren Wert für "$vtxt" erfassen?
    #j: Ja
        ~Set dummy=$validx; dummy+=1
        ~Include VWOutput varidx=$varidx; validx=$dummy; kbw=$kbw
    #n: Nein
```

Die Ausgangsdatei der Vorlage ruft zuerst eine Funktion für die Eingabe auf und erzeugt im Anschluss die Ausgabe der Kombinationen mit einer anderen Funktion. Die Funktion für die Eingabe einer Variation ruft ihrerseits wieder eine Funktion für die Eingabe eines Variationswertes auf. Beide Funktionen fragen am Ende nach, ob es weitere Variationen, bzw. Variationswerte gibt und rufen sich in diesem Fall mit einem hochgezählten Index rekursiv auf (vgl. auch Abschnitt 2.17).

Die Ausgabe der Kombinationen funktioniert mit einer einzigen Funktion, die sich mit Hilfe der schon bei der Eingabe beantworteten Fragen wahrweise mit erhöhtem Index für die Variation oder den Variationswert so selbst aufruft, dass alle Kombinationen durchlaufen werden.

Anstatt alle Kombinationen auszugeben könnte man mit Hilfe von Zufallswerten und If-Bedingungen auch eine Teilmenge bestimmen und diese für zufällige Stichproben verwenden (vgl. Abschnitt 2.28 und 2.34). Relevanz oder Testergebnis für eine Kombination könnte auch direkt durch die Vorlage vom Benutzer abgefragt werden.