Scripting Unterstützung
Dokumentation

Programmierer-Referenz

http://www.combit.net

Inhalt

1 Einführung

Ihnen steht mit dem Scripting in List & Label eine mächtige Erweiterungsmöglichkeit zur Verfügung, die den Zugriff auf die Variablen, Felder und mehr erlaubt. Mit Hilfe von Scripten können Sie diese ansprechen und somit viele Zusatzfunktionen komfortabel in einer Sprache Ihrer Wahl realisieren.

> Hinweis: Ein Script ist eine Abfolge von Befehlen, die bei der Ausführung sequenziell abgearbeitet werden. Die Befehle entstammen dabei dem "Wortschatz" einer bestimmten Scriptsprache. Dieser Befehlssatz bestimmt, welche Möglichkeiten die Sprache bietet und wie ein Script aufgebaut sein muss.

Scripts sind in der Regel nicht allzu umfangreich und führen schon mit wenigen Befehlen zu beachtlichen Leistungen. Ein durchschnittliches Script umfasst vielleicht 20 bis 40 Zeilen Befehle. Nicht zuletzt aus diesen Gründen sind Scriptsprachen meist sehr leicht zu erlernen.

Obwohl oberflächlich sehr ähnlich, gibt es doch eine ganze Reihe von entscheidenden Unterschieden zwischen Scripts und ausführbaren Programmen.

So sind Scripts beispielsweise nicht selbst lauffähig, sondern benötigen immer eine Umgebung, in der sie ablaufen. Diese sogenannten Hosts sind für die Verwaltung der Skripte verantwortlich und erweitern die Möglichkeiten der jeweiligen Sprache meist in Form von zusätzlichen Objekten. In diesem Fall ist List & Label der Host. Mittels des bereitgestellten Frameworks steht dem Anwender eine mächtige Schnittstelle zur Erweiterung der Funktionalität des Formeleditors zur Verfügung. Da List & Label-Skripte gewöhnlich innerhalb der Druckschleife häufig durchlaufen werden, dürfen diese selbstverständlich keinerlei GUI Elemente enthalten. Aus dem gleichen Grund sollten sie innerhalb eines überschaubaren Zeitrahmens ausgeführt werden können.

1.1 Welche Scriptsprachen werden unterstützt?

Grundsätzlich werden theoretisch alle Scriptsprachen des Windows Scripting Host unterstützt. Am weitesten verbreitet sind jedoch VBScript und JScript, die direkt vom Hersteller Microsoft angeboten werden. Es gibt aber auch andere Umsetzungen wie z.B. Python. Die Auswahl der zu verwendenden Sprache erfolgt mittels Parameterstring beim Aufruf der Skriptfunktionen.

> Hinweis: VBScript und JScript sind üblicherweise schon auf Ihrem System installiert. Falls nicht oder falls andere Sprachen verwendet werden sollen, müssen diese vom jeweiligen Hersteller bezogen werden und entsprechend seinen Angaben auf dem System installiert werden.

Neben den klassischen Scriptsprachen des Windows Scripting Host wird auch C# als Scriptsprache unterstützt.

Bei C# Scripten steht Ihnen der volle Funktionsumfang des .NET Frameworks 4.0 zur Verfügung, weshalb für das Ausführen von C# Scripten auch mindestens dieses oder ein neueres .NET Framework installiert sein muss.

Da C# Scripte vor der Benutzung kompiliert werden müssen und dies unter Umständen länger dauern kann als das Ausführen des Scriptes selbst, werden die 30 neuesten kompilierten Scripte für ein erneutes Ausführen zwischengespeichert. Die entsprechenden Dateien befinden sich im Ordner "%temp%\combitCSharpScriptCacheLL21\". Dort gibt es zum einen die Datei "combitCSharpScriptCache.cache", welche Informationen über die gespeicherten Scripte enthält, und zum anderen für jedes Script einen Ordner mit einem Namen in der Form "combitCSharpScript_[GUID]" (z.B. "combitCSharpScript_e9527e037aa149f3ba79bd408a8232db"). In diesem Ordner befinden sich jeweils alle vom Script benutzten .dll-Dateien, Debug-Informationen und das Script selbst (ebenfalls in Form einer .dll-Datei).

Wir empfehlen zudem unter Windows XP aufgrund diverser unterschiedlicher Systemstände (Windows Updates, Service Packs) Skripte auf dem Ziel-System zu testen.

1.2 Wie und wo lassen sich Skripte integrieren?

Die Integration von Skripten ist bequem über den in List & Label integrierten Formeleditor möglich. Der tatsächliche Scriptcode kann dabei entweder direkter eingebetteter Bestandteil einer Formel sein, oder alternativ kann mittels der LoadFile$ Designerfunktion auch externer Code referenziert werden. Desweiteren kann innerhalb des Codes zusätzlich mittels include Anweisungen weiterer externer Code eingebunden werden. Insbesondere bei größeren Scripts ist hier die Verwendung externer Textdateien die bessere Wahl, da diese eine leichte Wiederverwertbarkeit an anderer Stelle erlaubt.

Es werden folgende Designerfunktionen zur Verfügung gestellt:

Script$: Gibt das Ergebnis eines Scriptes als Zeichenkette zurück

ScriptBool: Gibt das Ergebnis eines Scriptes als Boolean zurück

ScriptDate: Gibt das Ergebnis eines Scriptes als Datum zurück

ScriptVal: Gibt das Ergebnis eines Scriptes als Zahl zurück

1.3 Support zu Scripting-Funktionalitäten

Die Möglichkeiten der Scripting-Technologie sind sehr weitreichend und deren Beschreibung daher sicherlich Stoff für ein eigenes Buch. Wir bitten um Ihr Verständnis, dass wir keine Beschreibung der verwendeten Scriptsprachen liefern können. Entsprechende Werke finden Sie im Buchprogramm vieler größerer Fachverlage.

Eine ausführliche Referenz zu VBScript und JScript bietet Microsoft im Internet an: http://www.microsoft.com/germany/technet/scriptcenter/.

Selbstverständlich wollen wir Ihnen im Rahmen unseres Supports bei Ihren Fragen und Wünschen gerne mit Rat und Tat zur Seite stehen, um Ihnen einen optimalen Einsatz unseres Produktes zu ermöglichen. Wir möchten jedoch um Verständnis bitten, dass wir nur auf Fragen zum Objektmodell selbst, nicht aber zu Modellen anderer Produkte oder zu den Scriptsprachen selbst eingehen können.

2 Präprozessor und Optionen

2.1 Aktivieren der Scriptengine

Standardmäßig ist die Scriptengine aus Sicherheitsgründen nicht aktiv, da hiermit dem Benutzer Möglichkeiten geboten werden im Kontext des aktuellen Anwendungsbenutzers Systemfunktionen über die Scriptsprache aufzurufen. Aus diesem Grund muss die Scriptengine zunächst aktiviert werden. Es stehen hierfür drei Optionen zur Verfügung.

Das allgemeine Scripting aktivieren (default false)

LISetOption(hJob, LL_OPTION_SCRIPTENGINE_ENABLED, true);

Optional kann ein benutzerdefinierter Timeout für die maximale Laufzeit eines Scriptes gesetzt werden (Default 10000ms). Ein Script mit längerer Laufzeit wird von der Umgebung abgebrochen. Bei C# Scripting ist hier mit zu geringem Timeout Vorsicht geboten, da eventuell anfallende Compile-Zeit zur Ausführungszeit zählt.

LISetOption(hJob, LL_OPTION_SCRIPTENGINE_TIMEOUTMS, 15000);

Optional kann der Formeleditor so eingestelllt werden, dass der Ausdruck bei jedem Tastendruck in Echtzeit ausgeführt wird. Da dies jedoch das System sehr belasten kann und je nach Scriptsprache und Inhalt problematisch sein könnte, ist hier der Default ebenfalls false.

LISetOption(hJob, LL_OPTION_SCRIPTENGINE_AUTOEXECUTE, true);

2.2 Allgemein für alle Sprachen

Alle Anweisungen für den Präprozessor wie Pragmas, Optionen, Includes müssen stets in eine eigene Zeile gesetzt werden, damit diese korrekt behandelt werden können. Führende Leerzeichen und Tabulatoren werden dabei ignoriert. Es ist jedoch möglich diese ad-hoc auszukommentieren.

Kommentarzeichen für alle Präprozessoranweisungen ist dabei // (analog C/C++/C#). Mehrzeilige Kommentarblöcke werden im Präprozessor nicht unterstützt.

Da Formelparameter innerhalb eines List & Label Ausdrucks entweder durch ' oder " gekennzeichnet werden, ist deren Verwendung innerhalb eines Skriptes eingeschränkt, wenn der Quelltext direkt in die Formel eingebettet wird. Man ist dann innerhalb des Quelltextes auf die Verwendung des jeweils bisher ungenutzten Symbols beschränkt. Wird der Quelltext jedoch stattdessen mittels der LoadFile$ Routine aus einer externen Datei geladen, so besteht diese Einschränkung natürlich nicht.

2.2.1 Auswahl der Scriptsprache

Über den Befehl

```
<!--#language="[Scriptsprache]"-->
```

kann die Scriptsprache zusätzlich innerhalb des Codes explizit gesetzt werden. Die Verwendung ist optional. Ist eine Angabe vorhanden, wird jedoch lediglich geprüft und gewarnt, falls unterschiedliche Sprachen vermischt werden. Die eigentliche Auswahl der Sprache erfolgt über den entsprechenden Parameter des Scriptaufrufes. Mögliche Werte sind die für Script$ benötigten Identifier wie z.B. "CSharpScript", "VBScript" oder "JScript".

2.2.2 Scripte in Scripten einbinden

Bei häufig benötigten Funktionen bietet es sich an, diese zentral abzulegen, so dass sich eventuell notwendige Änderungen auf alle darauf basierenden Scripts auswirken. Hierzu wird die Einbindung von Scripts über eine spezielle Anweisung der folgenden Form unterstützt:

```
<!--#include file="c:\scripts\include.vbs"-->
```

Die Anweisung wird dabei durch den kompletten Inhalt der angegebenen Datei ersetzt. Include Anweisungen wie alle anderen Pragmas und Optionen müssen stets in eine eigene Zeile gesetzt werden, damit sie vom Präprozessor korrekt behandelt werden können.

> Hinweis: Alle auf diese Weise eingebundenen Scripts müssen die gleiche Scriptsprache verwenden wie das Hauptscript. Eine Mischung von mehreren Sprachen ist nicht möglich und führt zu Syntaxfehlern.

Sofern Sie Ihre Scripts unterhalb des Programmverzeichnisses abgelegt haben, können Sie statt einer festen Angabe des Verzeichnisses die %APPDIR% Variable verwenden:

```
<!--#include file="%APPDIR%\include.vbs"-->
```

2.3 C# Spezifisch

2.3.1 Protokollierung [C#]

Um die Protokollierung eines Scriptes zu aktivieren, muss folgende Anweisung enthalten sein:

```
<!--#pragma forcelogging-->
```

Die Protokollausgaben erfolgen in die Datei "%temp%\combitCSharpScript.log". Protokollierung kann zeitintensiv sein und sollte daher standardmäßig nicht aktiviert werden.

2.3.2 Debugmodus [C#]

Enthält ein Script die Anweisung

```
<!--#pragma debugmode-->
```

dann wird zu Beginn des Scripts das Starten eines Debuggers ausgelöst (sofern auf Ihrem System ein solcher installiert ist), mit dem Sie das Script Schritt für Schritt auf Fehler überprüfen können und zum anderen enthalten durch Ausnahmen ausgelöste Fehlertexte mehr Informationen und Zeilennummern.

2.3.3 Hinzufügen von Referenzen [C#]

Über den Befehl

```
<!--#include ref="[Dateipfad]"-->
```

können einem Script externe Referenzen / Komponenten, wie z.B. die Windows Forms Bibliothek von Microsoft (System.Windows.Forms.dll), hinzugefügt werden.

Geben Sie keinen Pfad, sondern nur einen Dateinamen an, so wird versucht die Referenz aus dem "Global Assembly Cache" zu laden. Bei Angabe eines kompletten Pfads wird eine Kopie der Datei in einem temporären Ordner angelegt und von dort geladen.

> Hinweis: Dieser Befehl muss im Script nach den für alle Scriptsprachen geltenden Präprozessor-Direktiven erfolgen.

2.3.4 Hinzufügen von Namespaces [C#]

Über den Befehl

```
<!--#include using="[Namespace]"-->
```

können using-Anweisungen zum Script hinzugefügt werden. Dies hat den Vorteil, dass Namespace-Namen nicht immer explizit angegeben werden müssen.

Anstatt eines Aufrufs von "System.Collections.Generic.List<String> obj;" für jede einzelne Liste würde der Aufruf nach dem Hinzufügen von "System.Collections.Generic" als using-Anweisung nur noch "List<String> obj;" lauten.

> Hinweis: Dieser Befehl muss im Script nach den für alle Scriptsprachen geltenden Präprozessor-Direktiven erfolgen.

3 Kurzreferenz und Verwendungsbeispiele

Ein Skript wird mittels der Funktion Script$(<Sprache>, <Code>, <opt:Funktion>) aufgerufen und liefert eine Zeichenkette als Ergebnis zurück. Alternative Formen ScriptVal, ScriptBool und ScriptDate funktionieren bis auf den Rückgabetyp analog.

Der erste Parameter bestimmt die zu verwendende Skriptsprache. Unterstützt werden primär CSharpScript sowie VBScript und JScript.

Der zweite Parameter enthält den auszuführenden Scriptcode.

Der dritte Parameter definiert unter VBScript das Ergebnis der Rückgabe, er enthält entweder den Namen der auszuführenden Funktion/Methode oder einen Variablennamen. Für C# wird dieser dritte Parameter ignoriert. Die Rückgabe von Werten erfolgt dort direkt über Zuweisung der Variable WScript.Result.

3.1 Exemplarische Aufrufe

Beispiele für C#:

```
Script$('CSharpScript',' WScript.Result= "Sprache: " +
Report.Variable("LL.CurrentLanguage"); ')

Script$('CSharpScript', LoadFile$(ProjectPath$(false) + "Script.cs"))
```

Als weitere Referenz ist in den erweiterten Beispielen der Beispielanwendung das Projekt "Bestellliste mit Scripting.lsr“ bzw. "Order list with scripting.srt" enthalten.

Beispiele für VBScript:

```
Script$('VBScript',' RetVal= "Sprache: " + Report.Variable("LL.CurrentLanguage") ',
'RetVal')

Script$('VBScript', LoadFile$(ProjectPath$(false) + "Script.vbs"), RetVal)
```

3.2 Allgemeines Objektmodell

Innerhalb der Skripte kann auf bereitgestellte Variablen und Methoden des List & Label Hosts zugegriffen werden.

3.2.1 Report Objekt

```
Methode Report.Variable(<string>)          // Zugriff auf LL Variable, read only

Methode Report.Field(<string>)             // Zugriff auf LL Felder, read only
```

Beim Zugriff auf Variablen und Felder ist auf den aktuellen Kontext zu achten. Es kann natürlich nur auf die im aktuellen Kontext auch tatsächlich angemeldeten Variablen und Felder zugegriffen werden.

Beispiel:

```
var s = Report.Variable("LL.CurrentContainer");

var s = Report.Field("Orders.CustomerID");

Methode Report.SetVar(<string>, <value>)       // Aufruf der Designerfunktion SetVar

Methode Report.GetVar(<string>, <value>)       // Aufruf der Designerfunktion GetVar
```

Mittels der Designerfunktionen SetVar/GetVar können während des Drucks indirekt Zwischenergebnisse von einem Skript zum nächsten weitergereicht werden. Allerdings ist hier die Reihenfolge der Aufrufe (Spalten) natürlich entscheidend. Siehe auch Dokumentation SetVar/GetVar.

Beispiel:

```
Skript1 -> Report.SetVar("ResultTmp", value);

Skript2 -> var StartValue= Report.GetVar("ResultTmp");
```

3.2.2 WScript Objekt

```
Constant WScript.FullName          // Enthält vollständigen Namen der Applikation

Constant WScript.Name              // Enthält Namen der Applikation

Constant WScript.Path              // Enthält Pfad zur Applikation

Constant WScript.Version           // Enthält interne Versionsnummer
```

Die obigen Konstanten sind für den direkten Zugriff im Skript verfügbar.

Beispiel:

```
var MyFilePath= WScript.Path + "MyFile.txt";

Variable WScript.Result
```

Eine besondere Bedeutung kommt bei C# Scripten der Variable WScript.Result zu, da sie als Rückgabewert für sämtliche Scripte dient. Im Gegensatz zu z.B. VBScript ist dieser Rückgabewert fest vorgegeben und sollte innerhalb des Scripts mindestens einmal zugewiesen werden. Die zuletzt vorgenommene Zuweisung definiert das Ergebnis.

Bei anderen Scriptsprachen ist diese Variable nicht definiert.