Einführung in ZemprosysScript

Wenn mit Hilfe der Entwicklungsumgebung ZDS möglich ist, Applikationen rapide, visuell und ohne Schreibung eines Codes zu entwickeln, stellt sich die Frage, warum man noch eine Skriptsprache braucht, zumal auch zahlreiche Skript-Technologien existieren. Die Gründe für eine eigene Skriptsprache sind unterschiedlich: Mit einer eigenen Skript-Technologie sollte es Wege für Open Source Projekte eröffnet werden, da in diesem Fall freiwillige Entwickler die Mittel hätten, - auch ohne die Entwicklungsumgebung - eigene VPPT-Applikationen zu erstellen. Wenn man in Betracht zieht, dass ZCI (das Laufzeitprogramm) kostenlos vertrieben wird, ist die Einführung einer eigenen Skriptsprache eine zukunftsorientierte strategische Entscheidung für eine solide Etablierung von Zemprosys auf dem Softwaremarkt. Der andere Grund, warum wir auf **ZemoprosysScript** beharren, ist durch die grundlegenden Aspekte von **Zemprosys NetEngine** gegeben, weil diese Web-Plattform mit einem Datenbanksystem verknüpft ist, dessen Tabellen die VPPT-Applikationen in Textdateiformat (Skript-Applikationen) verwalten.

(1) Das Wesentliche von ZemprosysScript

ZemprosysScript ist eine Sprache, deren Grundstruktur und Syntax sehr einfach sind. Die Sprache ist komponentenbasiert und kennt bis dato meistens nur die Zuweisung als Anweisung, wobei Komponenteneigenschaften Werte zugewiesen werden. Die Werte, die zugewiesen werden können, sind Zahlen, String, Farbenwerte, Komponentenzustände usw.
Wenn man sich nur auf diese einfache Anweisung (Zuweisung) beschränkt, sollte man eigentlich nicht von einer Skriptsprache reden, da beispielsweise bedingte Anweisungen, Switch-Anweisungen, while-Schleife, do-Schleife oder for-Schleife direkt nicht formuliert werden können. Ebenfalls sind direkte Funktions- und Prozeduraufrufe nicht möglich (Funktionen und Prozeduren können nur indirekt durch eine Zuweisung aktiviert werden). Dennoch haben wir hier mit Programmierung zu tun: Aufgrund der komponentenbasierten Sprache richtet sich der Entwickler an den Anwendungsbereichen von Komponenten, um Skript-Applikationen bzw. VPPT-Applikationen zu erstellen. Jede Softwarekomponente verfügt über ausreichende Funktionsmöglichkeiten, die vom Entwickler genutzt werden können. Was vom Entwickler als primäre Voraussetzung verlangt wird, ist die Komponentenbibliothek von Zemprosys umfangreich zu kennen, indem der Entwickler wissen sollte, wie Eigenschaften und Methoden auf Komponenten wirken.

In der Regel schreibt man unter Anleitung von **ZemporsysDecriptor** Skript-Applikationen.

(2) Struktureller Aufbau einer Skript-Applikation

Auch wenn wir eben nur von einfacher Zuweisung als einzige wirkliche Anweisung geredet haben, verfügt eine VPPT-Applikation (in Skript-Form) über weitere Anweisungen (**generische Anweisungen**), die dazu dienen, die Applikation während des Ladenvorgangs zunächst ohne Applikationslogik zu rekonstruieren. Unter anderem werden Komponentenreferenzen gesetzt. Die folgende Zeilen spezifizieren den allgemeinen strukturellen Aufbau einer VPPT-Applikation in Skript-Form:

```
BeginInterface
        Zuweisung...
        Zuweisung...
        ...
EndOfInterface

BeginDeclaration
        Class = ObjectName: MyIden
        EndOfClass

        Class = ObjectName: MyIden
                Child = A
```

```
                Child = B
                Child = C
        EndOfClass

        Class = ObjectName: MyIden
        EndOfClass
EndDeclaration

BeginObject
        <MyIden> = A
        <ObjectName> = B
        <AlliasName> = C
        Zuweisung...
        Zuweisung...
        Zuweisung...

        BeginStructure: Name
        EndOfStructure

        ColorAlpha: Name
        EndOfList

        StructureAsFont: Name
        EndOfStructure

        StructureAsBrush: Name
        EndOfStructure

        StructureAsBorder: Name
        EndOfStructure

        StructureAsColor: Name
        EndOfStructure

        StructureAsImage: Name
        EndOfStructure

        StructureAsRelation: Name
        EndOfStructure

        BeginSimpleList: Name
        EndOfList

        BeginComplexList: Name
        EndOfList

        BeginCollection: Name
        EndOfCollection

EndObject

BeginObject
EndObject

BeginObject
EndObject
...
```

Beim vorliegenden Dokument präsentieren wir nur ganz kurz den strukturellen Aufbau einer Skript-Applikation. Eine ausführliche Beschreibung wird zusammen mit ZemprosysDescriptor auf einem anderen Dokument vorgenommen.

BeginInterface ... EndOfInterface
Definiert einen Block von Zuweisungen, die zur allgemeinen Initialisierung der Skript-Applikation dienen. Beispielweise kann die Standardüberschrift (Titel) oder die Hintergrundfarbe der Applikationen in innerhalb dieses Blocks zugewiesen werden.

BeginDeclaration ... EndDeclaration
Dieser Block ist für die Durchführung von Deklarationen vorgesehen. Alle Applikation-Komponenten werden hier registriert und zwar mit Hilfe einer Liste von Class...EndOfClass-Anweisungen. Sollen beispielsweise 5 Komponente verwendet werden, müssen daher auch 5 Class...EndOfClass-Anweisungen geschrieben werden.

Class ... EndOfClass
Eine Komponente wird für eine Applikation registriert, indem die Anweisung Class...EndOfClass geschrieben wird. Wie der strukturelle Aufbau es zeigt, besitzt diese Anweisung zwei wichtigen Parameter: **ObjectName** gibt den Name der Komponente an und **MyIden** die Identifikation der Komponenten. Der Komponentenname ist freiwillig zu bestimmen, wie z.B. „MeineListe“ und muss innerhalb der Applikation **eindeutig** sein. Die Identifikation MyIden ist eine Zahl, die auf eine kompilierte bzw. binäre Komponente der Komponentenbibliothek verweist. Mit Hilfe von ZemprosysDescriptor kann diese Zahl unkompliziert und anschaulich ermittelt werden. Daher kann MyIden nicht einfach so festgelegt werden (diese Zahl muss in der Zemprosys-Architektur existieren und wird vom Komponentenentwickler festgelegt)

Sofern eine Komponente in der Lage ist, andere Komponenten aufzunehmen - es handelt sich hier um Container - und aufnehmen sollte, wird das Schlüsselwort **Child** innerhalb der Class...EndOfClass-Anweisung verwendet. Durch Schreiben von Child = A, wobei A den Name einer deklarierten Komponente repräsentiert, spezifiziert man, dass die Komponenten A als Kind eine andere Komponente registriert werden muss. Man muss allerdings darauf achten, dass die gleiche Zuweisung Child = A nicht an mehreren Stellen vorkommt und zugleich dass die Komponente A tatsächlich auch deklariert ist.

Mit dem Schlüsselwort **Child** können Referenzen auf Referenzen gebildet werden und somit verschachtelte Container-Systeme generiert werden.

BeginObject ... EndObject
Jede deklarierte Komponente muss mit der BeginObject...EndObject-Anweisung strukturell und funktionell aufgebaut werden. Innerhalb des Blocks können einfache Zuweisungen, Struktur- und Kollektion-Anweisungen vorgenommen werden. Für jeden Aufbau einer deklarierten Komponente sind die Zuweisungen **<MyIden> = A** und **<ObjectName> = B** pflicht. A und B entsprechen den Werten, die als Parameter für Class...EndOfClass festgelegt sind.

BeginCollection ... EndOfCollection
Die Anweisung BeginCollection...EndCollection lässt das Schreiben einer Liste von Strukturen zu. Mit anderen Worten, innerhalb dieses Blocks dürfen nur Anweisungen der Form BeginStructure...EndOfStructure getragen werden. Sowohl homogene als auch heterogene Kollektionen können aufgebaut werden. Auch hier bestimmt der Komponententyp die Art der Kollektion, die aufgebaut werden kann.

BeginStructure ... EndOfStructure
Dient zur Beschreibung einer generischen Struktur bzw. eines Datensatzes. Innerhalb des Blocks dürfen nur einfache Zuweisungen geschrieben werden.

BeginSimpleList ... EndOfList
Die Anweisung BeginSimpleList ... EndOfList wird benutzt, um eine Liste von Werten in der Form **[] = Value[i]** zu schreiben, wobei i den Index eines Listenwertes repräsentiert.

BeginComplexList ... EndOfList
Ermöglicht ebenfalls das Schreiben einer Liste, wobei die einzelnen Zuweisungen innerhalb des Blocks die Form **Bez[i] = Value[i]** besitzen: Bez[i] gibt die Bezeichnung eines Listenelementes an und Value[i] dessen Wert.

ColorAlpha ... EndOfList
Diese Anweisung wird benutzt, um eine Farbenliste zu schreiben. Jeder Farbeneintrag wird durch eine Buchstabe identifiziert. Es können höchsten 26 Farbeneinträge spezifiziert werden und die zulässige Form der Zuweisung ist **[Buchstabe] = Value**, wie z.B. [A] = Color1 oder [B] = Color2

StructureAsFont ... EndOfStructure
Wird verwendet, um Schriftsinformationen (Schriftart, Schriftfarbe, Schriftgröße etc.) zu schreiben

StructureAsBrush ... EndOfStructure
Wird verwendet, um eine Schattierungsmuster zu schreiben.

StructureAsBorder ... EndOfStructure
Wird verwendet, um eine Rahmenart zu spezifizieren.

StructureAsColor ... EndOfStructure
Wird benutzt, um einen Datensatz bestehend aus Farben zu spezifizieren.

StructureAsImage ... EndOfStructure
Die Anweisung StructureAsImage ... EndOfStructure lässt das Schreiben von Bilderinformationen als Datensatz zu.

StructureAsRelation ... EndOfStructure
Diese Anweisung wird eingesetzt, um eine graphische Relation zu spezifizieren.

(3) Hinweise

- Eine VPPT-Applikation kann als eine Tabelle mit einer Spalte verstanden werden, wobei die Struktur jeder Tabellenzeile durch eine deklarierte Komponente bestimmt wird. Die Reihenfolge der Class...EndOfClass-Anweisungen ist daher sehr relevant.

- Für den strukturellen und funktionellen Aufbau einer Skript-Applikationen mit Hilfe der Anweisung BeginObject ... EndOfObject verfügt jeder Komponententyp über eine **Skript-Vorgabe**. Die Aufgabe eines VPPT-Applikation-Entwicklers besteht in diesem Zusammenhang meistens darin, die Skript-Vorgabe nach Anforderungen anzupassen. Der Inhalt jeder Skript-Vorgabe ist aus der Beschreibung von ZemprosysDecriptor zu entnehmen.

- Über die Entwicklungsumgebung ZDS können VPPT-Applikationen erzeugt werden und in Skript-Form abgelegt werden. Skript-Applikationen wiederum können geladen und visuell programmiert werden.

- Die Entwicklung der Skript-Technologie steht noch in ihrer Frühphase. Sicherlich wird der strukturelle Aufbau einer Skript-Applikation in Zukunft mit weiteren **generischen Anweisungen** unterstützt.