Gupta Team Developer CDK – Nützlich mit Tücken

Viele erfahrene Team-Developer-Entwickler kennen das CDK (Component Development Kit), eine API zum Zugriff auf Team-Developer-Programme bzw. auf deren Quellcode sicherlich bereits. Manche haben das CDK vermutlich auch schon selbst für unterschiedlichste Zwecke genutzt. Auch in unseren Projekten kommt es häufig vor allem zur Automatisierung verschiedener Aufgaben zum Einsatz. Falls Sie für sich noch kein Einsatzszenario finden konnten, hier ein paar produktiv im Einsatz befindliche Beispiele aus unseren aktuellen Projekten:

- Generierung von Projektdokumentationen (z.B. mittels unseres Tools „Dokumentations-Generator für SQLWindows“)
- Generierung von Datenbankzugriffsschichten aus Funktionalen Klassen (Stichwort UDV's)
- Generierung von Schnittstellen (insbesondere für Webservice-Zugriffe, die momentan mit Bordmitteln des Team Developers evtl. noch nicht möglich sind)
- Unterstützung der Migration von Anwendungen auf aktuelle Team-Developer-Versionen (z.B. lassen sich damit schnell alle Quellcodedateien in das Textformat überführen)
- Testautomatisierung
- Qualitätssicherung
- Vereinfachung von Routineaufgaben während der Programmierung als sogenanntes „User Tool“ (z.B. Vereinheitlichung aller Schriftarten oder -eigenschaften auf einer Maske)

Anhand dieser Beispiele kann man schon erkennen, dass das CDK nicht nur zum Auslesen des Quellcodes dient, sondern man damit auch ein Code verändern oder einen gänzlich neuen Code erstellen kann. Auch das Laufzeitverhalten lässt sich mittels der CDK-Schnittstelle überwachen.

So hilfreich und nützlich das CDK einerseits sein kann, so kostet es andererseits manchmal auch Nerven, wie der folgende Erfahrungsbericht zeigt:

Im Rahmen eines Kundenprojektes erarbeiteten wir gemeinsam mit unserem Kunden ein Toolset zur Qualitätssicherung der Gupta-Anwendungslandschaft. Dieses sollte aber nicht autark arbeiten, sondern sollte sich in die bestehenden Tools der vorhandenen IT-Plattform integrieren. Stichworte sind hier z.B. automatisierter Build mit Jenkins-Anbindung nebst automatisiertem Reporting.

Um diesen Anforderungen gerecht werden zu können, musste eine Anbindung des CDK an .Net-Programme (hier Programmiersprache C#) geschaffen werden. Nichts leichter als das, so dachten wir, schließlich gibt es ja neben der SAL- und C++-auch eine – zugegebenermaßen undokumentierte – .Net-Schnittstelle. Die Ernüchterung nach Einbinden der zugehörigen `CDKIxx.dll` erfolgte aber leider schnell – der Funktionsumfang ist hier sehr rudimentär. Nach Rücksprache mit den Entwicklern von Gupta wurde deutlich, dass diese Vorgehensweise kein gangbarer Weg ist, denn diese DLL ist lediglich für interne Zwecke z.B. für den .NET Explorer und Web Service Wizard gedacht und spezifisch auf diese DLL zugeschnitten.

Also, so fragten wir uns, doch den alten, mühsamen C++-Weg einschlagen? Dieser wird durch die DLL `lib/cdk.lib` repräsentiert, wird offiziell unterstützt und ist zudem dokumentiert. Wir haben uns dennoch dagegen entschieden, da diese DLL leider nicht auf dem aktuellsten Stand ist – es fehlen beispielsweise die Konstanten für einige der neueren Controls – und auch, weil uns das im Vergleich zu .Net-Sprachen „mühselige“ C++ zu aufwändig erschien.

Aber eine geniale Idee jagt bei uns die andere. Also der nächste Versuch: Da wir selbstverständlich auch eine Reihe von Team-Developer-.Net-Lizenzen unser Eigen

Gupta Team Developer CDK – Nützlich mit Tücken

nennen, konnten wir einfach die SAL-Schnittstelle nutzen (cdk.apl) und uns daraus eine .Net-DLL kompilieren. Leider fand der Compiler diese Idee nicht so gut wie wir selbst und unterband diesen Versuch jäh: Es ist offensichtlich nicht erlaubt, Team-Developer-eigene apl's nach .Net zu übersetzen.

Zum Glück gibt es eine weitere technische Umsetzungsvariante – diese ist vielleicht nicht so schön wie unsere ersten Ideen – dafür funktioniert sie (rückblickend) einwandfrei. Wir nutzen die gute, alte SAL-Schnittstelle mittels klassischer Gupta-Programmierung und erzeugen daraus ein Team-Developer-COM-Server-Objekt. Dieses ließ sich schnell und funktional bewerkstelligen und die Kommunikation mit der .Net-Komponente ist ebenfalls Standard.

Da wir dabei eine durchaus große Informationsmenge in komplexeren Strukturen benötigten, haben wir nun noch einen einfacheren Weg für die Datenübergabe als die klassischen Com-Objekte gesucht, um Aufwand, Codequalität und Wartbarkeit dauerhaft zu gewährleisten. Also wird von unserem Com-Server das Antwortobjekt in JSON-Notation als einzelne Zeichenkette (String) zurückgegeben. Einfach, aber effizient, da diese mit .Net-Mitteln leicht und sehr performant zu interpretieren ist. Daran, dass diese Zeichenketten dann leicht größer als 4 MB werden können – eine Standardgrenze im .Net – wurden wir bei der Gelegenheit erinnert. Ein, wenn man einmal auf die Ursache gekommen ist, aber ebenfalls leicht zu lösendes Problem …

MD Consulting & Informationsdienste GmbH

www.md-consulting.de

Michaelisstraße 13 a
99084 Erfurt
03 61 / 5 65 93-0

Berghamer Straße 14
85435 Erding
0 81 22 / 97 40-0

info@md-consulting.de