Ein Forth-Server-Interface

Dr.-Ing. Egmont Woitzel

FORTecH Software
Entwicklungsbüro Dr.-Ing. Egmont Woitzel
Joachim-Jungius-Straße 9 • 18059 Rostock
Telefon (0381) 4059-472 • Fax -471

Immer wieder wird das Problem diskutiert, wie man von möglichst beliebigen Programmiersystemen aus Forth beziehungsweise mit Hilfe von Forth implementierte Funktionen benutzen kann. Die fortschreitende Vernetzung und Dezentralisierung von Steuerungsintelligenz sowie die auch auf dem Schreibtisch zum Standard gewordene Benutzung von Multitaskumgebungen gibt diesem Problem unter dem Stichwort Komponentensoftware neue Aktualität.

Motivation

Das Softwarekarussell dreht sich immer schneller. Fast noch schneller als die Kosten für die Hardware sinken bzw. die verfügbaren Rechenleistungen steigen, nimmt die in einem einzelnen Softwaresystem implementierte Funktionalität zu. Die fortschreitende Verbindung der Rechner in Netzwerken führt zu einer weiteren Erhöhung der Komplexität.

Längst lassen sich wirklich große Softwarepakete nicht mehr in „einem Stück“ bauen. Offenbar sind hier die Grenzen der Objektorientierung auf der Ebene von Programm-Quelltext erreicht. Das neue Zauberwort der Softwareindustrie heißt Komponentensoftware, nichts anderes als Objektorientierung auf Niveau von separat übersetzten und in dieser Form wiederverwendbarer Softwarebausteine. Techniken wie OLE und OpenDoc wetteifern mit erheblichem Aufwand um die Definition universeller und plattformunabhängiger Interfaces.

Ebenso wie die Idee der Objektorientierung ist auch die der Komponenten-software nicht wirklich neu, sondern eher ein altbewährtes Grundprinzip modularer Softwareentwicklung. Nur haben heute Softwaresysteme eine solche Komplexität, daß ihre Implementation ohne die Wiederverwendung möglichst vieler Komponenten nicht mehr ökonomisch möglich ist.

Eine der vielleicht interessantesten Auswirkungen von Komponentensoftware auf den Softwaremarkt wird vermutlich darin bestehen, daß die Bedeutung der Programmiersprachen für die Wiederverwendbarkeit von Softwarekomponenten sinken wird. Komponenten können ohne Neuübersetzung binär zwischen verschiedenen Systemen und teilweise Plattformen ausgetauscht werden. Dies ist eine neue Chance für alternative Programmiersysteme wie Forth. Allerdings können diese nur dann wirklich von dieser Entwicklung profitieren, wenn sie tatsächlich anderen Komponenten Funktionalität zur Verfügung stellen können.

Interfaces

Betrachtet man die verschiedenen Möglichkeiten ein Interface zwischen zwei Softwarekomponenten zu definieren, wird man schnell einige wesentliche Forderungen erkennen. Zum einen sollte ein solches Interface möglichst effizient arbeiten, d. h. zum Aufrufzeitpunkt einer Funktion in der Serverkomponente sollten auf keiner Seite langwierige und nichtdeterministische Suchprozesse stattfinden müssen. Zum anderen sollte das Interface aber auch beide Komponenten nur möglichst lose miteinander koppeln, so daß z. B. bei einer Überarbeitung der Serverkomponente die Clientkomponente unverändert weiterbenutzt werden kann. Zudem soll ja auch eine Benutzung der Serverkomponente von einer in einer beliebigen Programmiersprache implementierten Clientenkomponente möglich sein. Zusätzlich sollte das Interface ein Minimum an Sicherheit gegen eine Fehlbedienung bieten, um die Stabilität des Gesamtsystems zu erhöhen. Da dies Überprüfungen zur Aufrufzeit erfordert, muß in einer konkreten Implementation immer ein Kompromiß zwischen Effizienz und Sicherheit gefunden werden.

Eine weiterer Punkt wird an die Wunschliste angefügt, wenn man die zur Implementation des Interfaces benutzte Software selbst wiederverwenden möchte. In diesem Fall sollte der Funktionsumfang der Serverkomponente, der von der Clientkomponente über das Interface gerufen werden soll, nicht explizit in der Interfacesoftware auftauchen. Möchte man die ganze Mächtigkeit von Forth über ein Interface erreichen, ist dies sogar zwingend notwendig. Aufgrund seiner Erweiterbarkeit ist die Anzahl der verfügbaren Funktionen in Forth ja keine statische Größe.

Verteilte Systeme

Die Komplexität des Problems wird weiter erhöht, wenn nicht nur auf einem einzelnen Rechner arbeitende Softwaresysteme betrachtet werden, sondern die miteinander kooperierenden Softwarekomponenten auf verschiedene Rechner verteilt sind. Im Bereich der Automation und Meßtechnik sind die beteiligten Systeme sind häufig grundverschieden und über diverse Kommunikationsmedien miteinander gekoppelt. Gerade bei der Inbetriebnahme derartiger Systeme kann Forth die Vorteile einer interaktiven Programmierumgebung zur Geltung bringen.

Ein Interface für eine verteiltes Softwaresystem sollte zusätzlich zu den oben aufgestellten Forderungen die Abwicklung der Kommunikation zwischen den beteiligten Rechnern übernehmen. Dabei sollten die Besonderheiten des benutzten Kommunikationskanals auf der Ebene des Interfaces selbst nicht sichtbar werden.

Forth-Interfaces

Die praktisch eingesetzten offenen Technologien zur Fernsteuerung von Forth-Systemen lassen sich grob in zwei Kategorien einteilen. Mit Abstand am häufigsten findet man Textinterpreter-basierende Lösungen. Hierbei wird der zu steuernden Komponente einfach Forth-Quelltext geschickt, der vom Text-

interpreter verarbeitet wird. Ein Beispiel für diese Art Interfacekonstruktion ist das Open Network Forth der Universität München. Textinterpreter-Interfaces sind sehr stabil und unkompliziert zu warten, da nur das Vorhandensein der benutzten Wortnamen gewährleistet werden muß. Allerdings haben sie insbesondere bei ressourcenkritischen Controlleranwendungen den Nachteil, daß ein vollständiger Textinterpreter in der Serverkomponente enthalten sein muß. Zudem sind Textinterpreter-Interfaces nicht laufzeiteffizient, da für jeden Token eine Suche im Wörterbuch durchgeführt wird. Insbesondere bei der Übertragung von Zahlen wird dies deutlich. Werden diese als Text übertragen, muß nicht nur zunächst das Wörterbuch durchkämmt werden, sondern auch noch der Prozeß der Konvertierung durchlaufen werden. ONF versieht Zahlen daher mit einem Präfix mit Eingabestromerwartung, dem die bereits binär konvertierte Zahl folgt.

Eine zweite Kategorie an Interfaces verschickt anstelle von Quelltext *execution token*, die in der Serverkomponente an den EXECUTE-Interpreter übergeben werden. Auf dieser Basis arbeitet zum Beispiel fieldFORTH im on-line-Betrieb mit dem Targetsystem zusammen. Derart konstruierte Interfaces benötigen auf Seite der Serverkomponente nur minimale Ressourcen und arbeiten sehr effizient. Allerdings benötigt die Clientkomponente Informationen über die den einzelnen Forth-Worten zugehörigen *execution token*, was die Softwarewartung größerer Systeme erheblich erschwert.

Der Gedanke liegt nahe, die Vorteile beider Ansätze für die Konstruktion von Applikationsinterfaces miteinander zu kombinieren. Die im folgenden als Dynamisches Token-Verfahren bezeichnete Technik ist ebenfalls nicht neu, wird aber im Unterschied zu den beiden oben beschriebenen Verfahren vergleichsweise selten verwendet.

Dynamischer Token-Code

Die Grundidee des Verfahrens besteht darin, daß während einer Konversation zwischen Client- und Serverkomponente ein im Server auszuführendes Wort nur ein einziges Mal als Textkette gesucht wird. Alle Aufrufe von Server-Worten durch den Client erfolgen über einen *service token*, der anhand des Wortnamens bestimmt wird. Dies kann der *execution token* sein, aus Gründen der Stabilität kann aber auch ein besser überprüfbarer Wert verwendet werden, z. B. ein Tabellenindex, der ohne großen Overhead auf seine Zulässigkeit getestet werden kann.

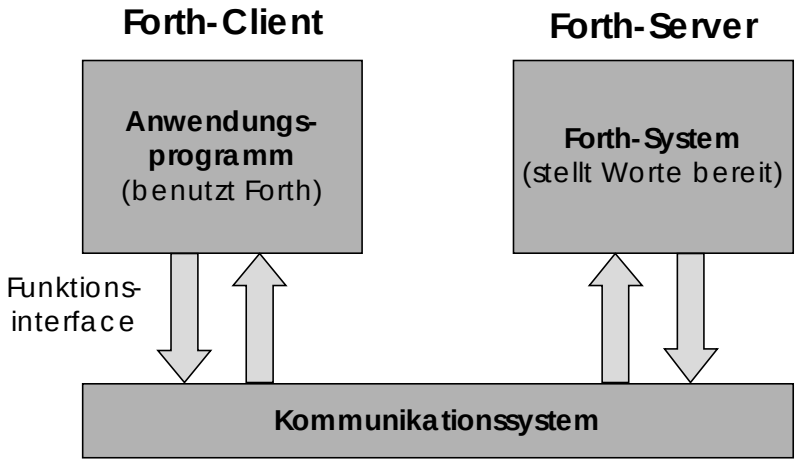
Ein Client muß bei Eröffnung der Konversation nur die Kenntnis über genau einen einzigen *service token* besitzen oder erhalten. Dieser muß derjenigen Funktion im Server zugeordnet sein, welche die Wandlung einer Zeichenkette in einen *service token* ausführt. Je nach Anforderung der Applikation wird der Client zum Zeitpunkt der Verbindungsaufnahme die *service token* aller in Frage kommenden Worte bestimmen oder dies erst beim jeweils ersten Aufruf eines Wortes tun. Die erste Strategie bietet sich an, wenn durch die Applikation ein genau bekannter Wortschatz benutzt wird. Trägt die Applikation dagegen eher den Charakter einer Kommando-Shell, wird man vermutlich die zweite Strategie verwenden. Der Ressourcenbedarf eines auf dynamischen Token basierenden Interfaces ist wegen der nötigen Zeichenketten und gegebenenfalls Tabellen zwar größer als der

eines EXECUTE-Interfaces aber immer noch wesentlich geringer als der eines Textinterpreter-Interfaces.

Auf dynamischen Token basierende Interfaces werden zum Beispiel innerhalb von fieldFORTH zur Steuerung des Targetservers durch die Shell benutzt. Ein anderes Beispiel stellt das Applikationsinterface der Echtzeit-Farbbildanalyse-Einheit EFA der graphikon GmbH Berlin dar. In beiden Fällen wird aber durch die Clientkomponente das Kommunikationsinterface direkt bedient, was einer einfachen Wiederverwendbarkeit im Wege steht.

Interfacearchitektur

Eine auf Komponenten basierende Client-Server-Architektur zur Benutzung von Forth-Worten läßt sich allgemein in untenstehender Form darstellen. Der Forth-Client wird mit Hilfe des Kommunikationssystems Wortaufrufe und Parameter an den Server übermitteln und von diesem auf Anforderung Ergebnisse zurückerhalten. Dabei wird er eines der diskutierten Interfaceverfahren, vorzugsweise jedoch das dynamische Token-Verfahren einsetzen.



Nimmt man die oben behandelte Forderung nach Verdeckung des Kommunikationsprotokolls durch das Interface hinzu und betrachtet Windows als Zielplattform für das Anwendungsprogramm, so bietet sich an, das Interface in eine separate DLL auszulagern, in der die Bedienung des konkreten Interfaces gekapselt wird. Sofern das obere Interface der DLL funktionell gleich bleibt, kann die Applikation dann ohne weiteres nur durch Austausch der DLL mit einem anders angeschlossenen Forth-System zusammenarbeiten.

DLL-Interface

Auf Ebene des DLL-Interfaces müssen drei wesentliche Aufgabengruppen wahrgenommen werden. Zum einen muß die Verbindung zwischen der Client- und Serverkomponente verwaltet werden, zum zweiten muß das Auflösen und Ausführen von Services organisiert werden und schließlich muß der Parametertransfer zwischen beiden Komponenten verwaltet werden. Das letztgenannte Problem verlangt dabei die am schwierigsten zu treffenden Entscheidungen. Programmierer, die Forth nicht direkt benutzen, aber

Funktionen einer Forth-Komponente verwenden sollen, haben oft Schwierigkeiten im Umgang mit der Forth-Stapel-Architektur. Daher findet man nicht selten Interfacedefinitionen, die jedes einzelne Forth-Wort mit Input- und Output-Parameter deklarieren, um den notwendigen Parametertransfer automatisch auszuführen. Das führt aber dazu, daß viele nur als Zwischenergebnisse benötigte Werte unnütz zwischen Client und Server hin und her wandern. Es ist daher weitaus effizienter, Parametertransfer und Wortaufrufe voneinander zu trennen und dem Applikationsprogrammierer die Forth-Stapel explizit zur Verfügung zu stellen. Im praktischen Einsatz zeigt sich, daß der durch comFORTH und fieldFORTH implementierte Kettenstapel den Transfer von Zeichenketten vereinfacht, da ja keine direkter Zugriff auf Speicherbereiche des Servers möglich sind. Ein weiterer Stapel für Fließpunktzahlen würde mit einem analogen Interface versehen.

Die Funktionen der DLL sollten den Windows-Konventionen folgend im Win16 als FAR PASCAL deklariert werden. Die untenstehende Tabelle gibt einen Überblick über die implementierten Funktionen. Um keine Konvertierungsprobleme zu provozieren wurden spezielle Typen für die Forth-Datenformate definiert. Zeichenketten werden in C-Konvention übergeben. Um prinzipiell eine parallele Konversation einer Anwendung mit mehreren Targetsystemen zu ermöglichen, wird die Clientkomponente von der DLL per *client identifier* CID, die jeweilige Serverkomponente durch einen *target identifier* TID identifiziert. Die für die Verbindungsaufnahme relevanten Informationen sind in einem Windows-Profile untergebracht.

Funktion	Beschreibung
Verbindungsmanagement	
`CID ForthInitialize()`	Client an DLL anmelden
`void ForthUninitialize(CID)`	Client an DLL abmelden
`TID ForthConnect(CID, char far *profile, char far *section)`	Verbindung zum Server herstellen (anhand Daten aus profile, [section])
`void ForthDisconnect(CID, TID)`	Verbindung zum Server beenden
Forth-Service-Management	
`SVID GetForthService(CID, TID, char far*)`	service token ermitteln
`void DoForthService(CID, TID, SVID)`	service token im Server ausführen
`void FlushForthServices(CID, TID)`	Service-Cache leeren
Parametertransfer	
`void PushInteger(CID, TID, FORTHINT)`	Integerwert auf Serverstapel legen
`void PushCharacter(CID, TID, FORTHCHAR)`	Zeichen auf Serverstapel legen
`void PushLong(CID, TID, FORTHLONG)`	Doppelinteger auf Serverstapel legen
`void PushString(CID, TID, char far*)`	Kette auf Serverkettenstapel legen
`FORTHINT PopInteger(CID, TID)`	Integerwert vom Serverstapel holen
`FORTHCHAR PopCharacter(CID, TID)`	Zeichen vom Serverstapel holen
`FORTHLONG PopLong(CID, TID)`	Doppelinteger vom Serverstapel holen
`UINT PopString(CID, TID, char far*, UINT)`	Kette vom Serverstapel holen

Client-Implementation in Forth

Verwendet man zur Implementation der Clientkomponente ebenfalls Forth, kann man das DLL-Interface relativ elegant unter einer Forth-angepaßten Syntax verstecken. Der zugehörige Quelltext ist im Anhang zu finden. Damit ist z. B. folgender Sitzungsablauf vorstellbar:

```
SERVICE: TCR CR          \ CR im Server
SERVICE: T". ".          \ ". im Server

C:\CFW\FDDECL.DLL        \ DLL-Name
C:\CFW\F2F.INI           \ Profile für Servicenamen
" TARGET"                \ Profile-Sektion
{FORTHSV                 \ Verbindungsaufbau

TCR                      \ CR im Server-Workspace
" Hello world!" "PUSH    \ Kette zum Server senden
T".                      \ und dort ausgeben
SYNC                     \ Server synchronisieren

}FORTHSV                 \ Verbindung abbauen und
                         \ DLL entladen
```

Von besonderem Interesse ist dabei die Implementation des Wortes SERVICE: die das dynamische Token-Protokoll transparent für den Benutzer abwickelt:

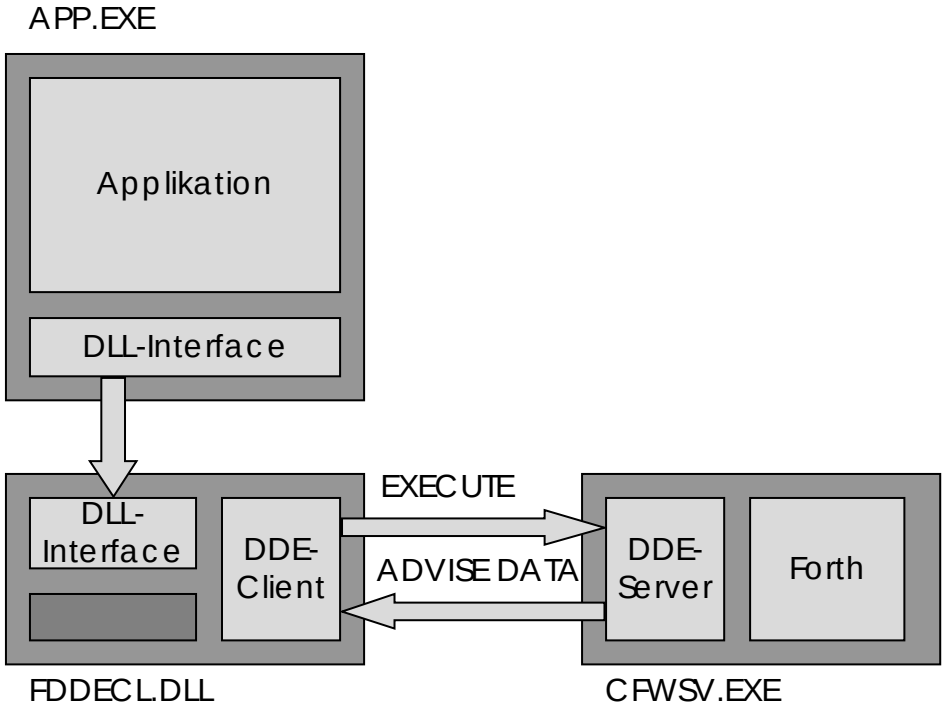
```
: SERVICE: ( ps: -- )( ib: name service )
     CREATE 0 ,          \ service id
     HERE SV^ @! ,       \ link service
     BL WORD "@ ",       \ service name
     DOES> ( ps: -- )
     DUP @ ?DUP 0=       \ svid lesen
     IF   \ erste Benutzung
          DUP 2 CELLS +
          "@             \ service name
          GETSERVICE     \ svid holen
          2DUP SWAP !    \ und merken
     THEN
     NIP DOSERVICE ;     \ svid ausführen
```

Es empfiehlt sich, die durch SERVICE: definierten Worte in einer verketteten Liste (Anker in der Variablen SV^) zusammenzufassen, da dadurch ein definiertes Rücksetzen aller aufgelöster Token vor einem Sitzungsstart möglich wird, wie die nachfolgende Definition von 0!SERVICES zeigt. Analog ist auch das geschlossene Bestimmen aller Token beim Applikationsstart möglich.

```
: 0!SERVICES ( ps: -- )( alle svid's rücksetzen )
     SV^                          \ Anker
     BEGIN  @ ?DUP                \ Ende?
     WHILE  0 OVER 1 CELLS - !    \ ausknipsen
     REPEAT ;
```

Modellimplementation

Das beschriebene Interface wurde anhand einer Modellimplementation getestet, die von der DLL aus eine DDE-Verbindung zu einem comFORTH-System aufbaut. Die durch die Serverkomponente auszuführenden Kommandos werden in Form von dynamisch bestimmten Token übertragen, die Indizes in eine serverseitige Tabelle darstellen. Zur Verbesserung des Durchsatzes werden so wenig DDE-Transaktionen wie möglich ausgeführt. Das bedeutet, daß Anweisungen innerhalb der DLL solange gesammelt werden, bis entweder eine explizite Synchronisierung erfolgt oder der Client einen Rückgabewert vom Server anfordert.



Die Kommandos werden mit Hilfe einer DDE-EXECUTE-Transaktion an den Server gereicht, Rückgabewerte vom Server über einen *hot link* und den entsprechenden DDE-ADVISE-DATA-Transaktionen zurückgegeben. Zur zeitlichen Entkopplung werden die Daten innerhalb der DLL in einem FIFO zwischengespeichert.

Ein Detailproblem stellte die Implementation des Forth-Service innerhalb von Windows dar. Um zu erlauben, daß das Ergebnis einer DDE-EXECUTE-Transaktion nicht stapelneutral sein muß, wurde die Interpretation der innerhalb der Transaktion übergebenen Liste von dynamischen Token aus dem DDE-Callback in die Message-Loop der Applikation verlegt. Allerdings kann so zu einem Zeitpunkt nur ein Client den Server benutzen.

Die hier vorgestellte Technik ist zur Definition eines effizienten und universellen Interfaces zu nahezu beliebigen Forth-Systemen geeignet und mit relativ wenig Aufwand auf der Serverseite implementierbar. In weiteren Arbeiten wird die Einbindung in den fieldFORTH-Nucleus folgen.

Programmlisting Forth-DLL-Interface

```
\ --- Globale Vereinbarungen

DLL: FORTHSV \ DLL-Deskriptor

0 VALUE cid  \ aktueller client
0 VALUE tid  \ aktuelles Target


\ --- Verbindungsaufnahme

0 1 FORTHSV DLLPROC: ForthInitialize
       ( ps: ===> cid )( an DLL anmelden )

1 0 FORTHSV DLLPROC: ForthUninitialize
       ( ps: cid ===> )( an DLL abmelden )

5 1 FORTHSV DLLPROC: ForthConnect
       ( ps: cid lp-Profile lp-Section ===> tid )
       ( mit Target entsprechend Profile verbinden )

2 0 FORTHSV DLLPROC: ForthDisconnect
       ( ps: cid tid ===> )( Target abmelden )


\ --- Dienstleistungen
@DOER STDDLLHOOK

MAKE STDDLLHOOK ( ps: dw ===> cid tid dw )
       cid tid 2SWAP ;

4 0 FORTHSV DLLPROC: PushString
       ( ps: lp-string ===> )( legt Kette auf Stapel )

MAKE STDDLLHOOK ( ps: w ===> cid tid w )
       cid tid ROT ;

EXPORT
3 0 FORTHSV DLLPROC: PUSH BINDTO: PushInteger
       ( ps: w ===> )( legt Integer auf Stapel )

EXPORT
3 0 FORTHSV DLLPROC: CPUSHBINDTO: PushCharacter
       ( ps: c ===> )( legt Zeichen auf Stapel )

EXPORT
3 0 FORTHSV DLLPROC: DOSERVICE   BINDTO: DoForthService
       ( ps: svid ===> )( führt Service aus )

MAKE STDDLLHOOK ( ps: ===> cid tid )
       cid tid ;

4 1 FORTHSV DLLPROC: GetForthService
       ( ps: lp-servicename ===> svid )
       ( beschafft svid für namen )

EXPORT
2 0 FORTHSV DLLPROC: SYNC BINDTO: FlushForthServices
       ( ps: ===> )( synchronisiert den Server )

EXPORT
2 1 FORTHSV DLLPROC: POP  BINDTO: PopInteger
       ( ps: ===> w )( Integer vom Stapel holen )

2 1 FORTHSV DLLPROC: PopCharacter
       ( ps: ===> c )( Zeichen vom Stapel holen )

3 2 FORTHSV DLLPROC: PopLong
       ( ps: ===> hi lo )( Doppelinteger vom Stapel holen )

MAKE STDDLLHOOK ( ps: lp w ===> cid tid lp w )
       >R cid tid 2SWAP R> ;

5 1 FORTHSV DLLPROC: PopString
       ( ps: lp-str u1 ===> u2 )( Kette holen )

MAKE STDDLLHOOK ( ps: lo hi ===> cid tid hi lo )
```

```
      cid tid 2SWAP SWAP  ;

EXPORT
4 0 FORTHSV DLLPROC: 2PUSHBINDTO: PushLong
      ( ps: dw ===> )( legt Doppelinteger auf Stapel )

!DOER STDDLLHOOK


\ --- Anpassungen

{EXPORT
: GETSERVICE ( ps: ===> svid )( "s: s ===> )( Service besorgen )
      LPCSTR GetForthService "DROP ;

: "PUSH ( ps: ===> )( "s: s ===> )( Kette auf Stapel legen )
      LPCSTR PushString "DROP ;

: 2POP ( ps: ===> dw )( holt Doppelwert vom Stapel )
      PopLong SWAP ;

: CPOP ( ps: ===> c )( holt Zeichen vom Stapel )
      PopCharacter SPLIT DROP ;

: "POP ( ps: ===> )( "s: ===> s )( Kette vom Stapel holen )
      255 "ALLOCATE LSTRDUP PopString "LEFT ;

: }FORTHSV ( ps: ===> )( schließt Verbindung zu Forth )
      tid
      IF    \ Verbindung existiert
            cid tid ForthDisconnect
            0 TO tid
      THEN
      cid ?DUP
      IF    \ Client ist angemeldet
            ForthUninitialize
            0 TO cid
      THEN
      FORTHSV @HDLL
      IF    \ DLL ist geladen
            FORTHSV }DLL
      THEN ;

: {FORTHSV ( ps: ===> ? )( "s: dll profile section ===> )
      ( öffnet Verbindung zu Forth entsprechend Strings )
      ( ?t, falls nicht erfolgreich )
      "ROT FORTHSV {DLL
      FORTHSV @HDLL 0=
      IF    \ DLL nicht geladen
            "DROP "DROP TRUE;
      THEN
      ForthInitialize ?DUP 0=
      IF    \ Initialisierung fehlgeschlagen
            FORTHSV }DLL
            "DROP "DROP TRUE;
      THEN
      DUP TO cid
      "SWAP 0 CHARAPP "SWAP 1 LSTRPICK DROP LPCSTR
      ForthConnect "DROP "DROP
      ?DUP 0=
      IF    \ Forth-Verbindung fehlgeschlagen
            cid ForthUninitialize
            0 TO cid
            FORTHSV }DLL
            TRUE;
      THEN
      TO tid FALSE ;

: 0!FORTHSV ( ps: ===> )( Kaltstartinitialisierung )
      FORTHSV 0!HDLL
      0 TO cid
      0 TO tid ;
```