uString

JEDNOTKA PRO DEKÓDOVÁNÍ INICIALIZAČNÍCH ŘETĚZCŮ A PRO PRÁCI S TEXTOVÝMI ŘETĚZCI

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 1.20

Informace v tomto dokumentu byly pečlivě zkontrolovány a SofCon věří, že jsou spolehlivé, přesto SofCon nenese odpovědnost za případné nepřesnosti nebo nesprávnosti zde uvedených informací.
SofCon negarantuje bezchybnost tohoto dokumentu ani programového vybavení, které je v tomto dokumentu popsané. Uživatel přebírá informace z tohoto dokumentu a odpovídající programové vybavení ve stavu, jak byly vytvořeny a sám je povinen provést validaci bezchybnosti produktu, který s použitím zde popsaného programového vybavení vytvořil.
SofCon si vyhrazuje právo změny obsahu tohoto dokumentu bez předchozího oznámení a nenese žádnou odpovědnost za důsledky, které z toho mohou vyplynout pro uživatele.

Datum vydání: 20.04.2004
Datum posledního uložení dokumentu: 20.04.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Upozornění:
V dokumentu použité názvy výrobků, firem apod. mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.XX	Wi		První vydání.
1.10	2.XX	Tu	20.05.2003	Úprava dokumentu dle ISO9000.
1.20	2.5X	Wil	20.04.2004	Odstranění direktiv překladu pro Ver6, tj. knihovna je určena pro Borland Pascal 7.0. Přidání funkcí: Enlarge, EnlargeLeft, EnlargeRight, SetLength, ReverseStr, WrapText, UpCaseEach1stChar.

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky pro dekódování inicializačních řetězců a pro práci s textovými řetězci.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu není potřeba číst žádný další manuál, ale je potřeba se orientovat v používání programového vybavení SofCon.

Popis formátu verze knihovny a souvisejících funkcí je popsán v manuálu LibVer.

Knihovna uString používá knihovnu NumToStr, která definuje různé funkce pro převody čísel a různých datových struktur na textové řetězce (string) a zpět.

2. Termíny a definice

Používané termíny a definice jsou popsány v samostatném dokumentu „Termíny a definice“.

3. Úvod

Jednotka uString definuje objekt **tCmd**, jehož instance slouží pro dekódování inicializačních stringů. Jednotka dále definuje několik samostatných procedur a funkcí, nezačleněných do objektu tCmd. Tyto procedury a funkce slouží pro obecné transformace typů na textové řetězce, pro prohledávání obsahu tabulky klíčových slov, pro úpravu inicializačních řetězců apod.

4. Popis jednoduchých konstant a typů

```
cVerNo = např. $0251; { BCD formát }
cVer   = např. '02.51,07.08.2003';
```

Konstanty čísla verze jednotky v BCD tvaru a v textové podobě včetně datumu změny.

```
ChBuf = array[0..10000]of char;
```

Typ **ChBuf** definuje pole znaků, na které se dá přetypovat libovolná proměnná typu string. V samotné jednotce se nevyužívá, slouží pro uživatele.

```
pChBuf =^ChBuf;
```

Typ **pChBuf** definuje typ ukazatele na pole znaků **ChBuf**.

```
tCmdString  = String[128];
pCmdString  = ^tCmdString;
tParamStr   = String;
pParamStr   = ^tParamStr;
tWordString = String[64];
tString12   = String[12];
```

Typy různě dlouhých textových řetězců, použitých v jednotce, pro úsporu paměti zásobníku.

Typ **tCmdString** definuje typ řetězce, který se používá pro příkazovou řádku DOSu.

Typ **tParamStr** definuje typ řetězce, který se používá pro dekódované řetězce instancí objektu tCmd.

Typ **tWordString** definuje typ řetězce, který se používá pro čtení jednotlivých slov z dekódovaného řetězce instancí objektu tCmd.

```
WordDelimiters: tCharSet = [#0..#255] -
                           ['0'..'9','A'..'Z','_','a'..'z'];
```

Typová konstanta definující množinu znaků používaných jako oddělovače slov.

5. Objekt tCmd

Objekt **tCmd** obsahuje metody, které slouží k dekódování inicializačních řetězců. Metody objektu umí dekódovaný řetězec rozložit na jednotlivá slova rozdělená volitelnými oddělovači a tato slova předávat aplikačnímu programu, který je může porovnávat se slovy v tabulce klíčových slov. Řetězec pro dekódování zadáme objektu tCmd parametrem metody **InitCmd**, nebo zavoláním metody **InitCmdPar**, která si ho přečte přímo z příkazové řádky DOSu. Z toho plyne, že po vytvoření instance objektu se musí zavolat některá z těchto dvou metod.

5.1. Položky

```
Delimiters : tCharSet;
```

Položka **Delimiters** určuje množinu znaků pro oddělení jednotlivých slov v dekódovaném řetězci.

```
CmdS       : pParamStr;
```

Položka **CmdS** definuje ukazatel na dekódovaný řetězec parametrů. Tento řetězec se po zavolání některé inicializační metody naplní a potom s ním ostatní metody vykonávají požadované operace.

```
CmdI       : Byte;
```

Položka **CmdI** definuje ukazovátko na jednotlivé znaky dekódovaného řetězce. Hodnota této položky určuje aktuální pozici v dekódovaném řetězci. Metody začínají svou činnost na celém slově za písmenem, na které ukazuje **CmdI**, a při ukončení své činnosti nastavují **CmdI** na první oddělovač za poslední slovo, které zpracovávaly. Tím je zajištěno postupné zpracování celého dekódovaného řetězce.

5.2. Metody

5.2.1. InitCmdPar

```
procedure InitCmdPar;
```

Procedura **InitCmdPar** převede řetězec z příkazové řádky DOSu do dekódovaného řetězce, ukazovátko **CmdI** nastaví na počátek tohoto řetězce a do množiny **Delimiters** přiřadí znak '=' a znaky z intervalu ordinálních hodnot 00h až 20h včetně.

5.2.2. InitCmd

```
procedure InitCmd(const S: tParamStr);
```

Procedura **InitCmd** se chová stejně jako **InitCmdPar**, ale dekódovaný řetězec naplní z předávaného parametru S.

5.2.3. ReadRest

```
function ReadRest: tParamStr;
```

Procedura **ReadRest** zkopíruje do své funkční hodnoty zbytek dekódovaného řetězce za ukazovátkem **CmdI**. Jako první slovo zkopíruje vždy celé slovo za oddělovačem. To znamená: Pokud **CmdI** ukazuje na oddělovač, funkce nalezne nejbližší další slovo a tím začne kopírování. Pokud **CmdI** ukazuje na začátek slova, funkce začne kopírování tímto slovem. Pokud **CmdI** ukazuje doprostřed slova, je toto slovo vynecháno a kopírování se začne nejbližším dalším slovem.

5.2.4. ReadWord

```
function ReadWord: tWordString;
```

Funkce **ReadWord** přečte z dekódovaného řetězce jedno celé slovo za **CmdI** od oddělovače k oddělovači. Pokud **CmdI** ukazuje doprostřed slova, je toto slovo vynecháno a kopírování se začne nejbližším dalším slovem.

5.2.5. ReadWordUpCase

```
function ReadWordUpCase: tWordString;
```

Funkce **ReadWordUpCase** provádí stejnou činnost jako funkce **ReadWord** s tím rozdílem, že písmena v navraceném slově převede na velká písmena. K tomu používá funkci **UpCaseStr**, která není součástí tohoto objektu.

5.2.6. ReadString

```
function ReadString: tCmdString;
```

Funkce **ReadString** přečte z dekódovaného řetězce část řetězce uzavřenou mezi apostrofy a nacházející se za **CmdI** a výsledek pak vrátí jako svou funkční hodnotu. Apostrofy samotné se do funkční hodnoty nepřenášejí.

5.2.7. ReadElement

```
function ReadElement:tCmdString;
```

Funkce **ReadElement** přečte z dekódovaného řetězce buď část řetězce uzavřenou mezi apostrofy či uvozovky, nebo jedno slovo (identifikátor) omezený oddělovači. Při čtení přeskočí případné komentáře uzavřené v lomených závorkách. Ohraničující apostrofy a uvozovky se nepřenášejí do funkční hodnoty a počáteční mezery za oddělovačem se vynechávají.

5.2.8. ReadElement1

```
function ReadElement1:tCmdString;
```

Funkce **ReadElement1** se chová stejně jako funkce **ReadElement**, ale přenáší do funkční hodnoty řetězce včetně ohraničujících apostrofů, uvozovek a mezer.

5.2.9. ReadLVal

```
procedure ReadLVal(var V:longint; var ErrFl:Boolean);
```

Procedura **ReadLVal** přečte jedno slovo z dekódovaného řetězce za **CmdI** a převede ho na celé číslo typu longint do "var" parametru V. Neúspěšnost převodu vrací v parametru ErrF, kde hodnota true značí neúspěšný převod.

5.2.10. ReadRVal

```
procedure ReadRVal(var V:real; var ErrFl:Boolean);
```

Procedura **ReadRVal** přečte jedno slovo z dekódovaného řetězce za **CmdI** a převede ho na reálné číslo typu real do "var" parametru V. Parametr ErrFl má stejný význam jako v metodě **ReadLVal**.

6. Procedury a funkce

Následující procedury a funkce nejsou součástí objektu tCmd. Jsou to převážně obecné konverzní funkce, funkce pro analýzu řetězce podle tabulky klíčových slov, pro různé zpracovávání řetězců apod.

Funkce **LStr**, **RStr** a **PtrStr** byly vytvořeny před knihovnou NumToStr, která definuje podobné funkce stejného významu, a proto jsou zde jen pro kompatibilitu se staršími aplikacemi. Je vhodné proto používat ekvivalentních funkcí z jednotky NumToStr.

6.1. LStr

```
function LStr(L: Longint): tWordString;
```

Funkce **Lstr** převádí číslo L typu Longint na textový řetězec.

6.2. RStr

```
function RStr(R: Real; A, B: Byte): tWordString;
```

Funkce **Rstr** převádí reálné číslo R na textový řetězec. Při převodu vypíše číslo R celkem A znaky, z toho je B desetinných míst. Chová se jako pascalská funkce **Str**, ale řetězec vrací jako svou funkční hodnotu.

6.3. PtrStr

```
function PtrStr(P: Pointer): tString12;
```

Funkce **PtrStr** převádí ukazatel P na řetězec formátu '$Segment:$Offset'. Výpis částí Segment a Offset je v jejich hexadecimální representaci.

Příklad výpisu : '$23A2:$12F8'.

6.4. UpCaseStr

```
function UpCaseStr(const S:String): String;
```

Funkce **UpCaseStr** převádí řetězec ASCII znaků S na velká písmena. Funkce mění znaky 'a' až 'z' na znaky 'A' až 'Z', ostatní nechává beze změny.

6.5. UpperCase

```
function UpperCase(const S:String): String;
```

Funkce **UpperCase** převádí řetězec ASCII znaků S na velká písmena. Funkce mění znaky 'a' až 'z' na znaky 'A' až 'Z', ostatní nechává beze změny. Funkce je identická s funkcí UpCaseStr. Pro budoucí alternativní možnost přechodu Vaší aplikace pod Delphi je vhodnější používat funkci UpperCase, kterou definují interní knihovny Delphi.

6.6. LowerCase

```
function LowerCase(const S:String): String;
```

Funkce **LowerCase** převádí řetězec ASCII znaků S na malá písmena. Funkce mění znaky ‘A’ až ‘Z’ na znaky ‘a’ až ‘z’, ostatní nechává beze změny. Jedná se o inverzní funkci k funkci UpperCase.

6.7. UpCaseEach1stChar

```
function UpCaseEach1stChar(const S : String) : String;
```

Funkce **UpCaseEach1stChar** převede zadaný řetězec znaků S (7-bit ASCII) na malá písmena kromě prvních znaků všech slov, která převede na velká. Výsledný řetězec vrátí jako svoji funkční hodnotu. Tj. ve výsledném řetězci všechna slova začínají velkými písmeny a zbytky slov jsou malými. Jako oddělovače slov používá množinu WordDelimiters.

6.8. Trim

```
function Trim(const S: string): string;
```

Funkce **Trim** odstraní z řetězce S všechny počáteční a koncové mezery a řídicí znaky (znaky, jejichž ASCII hodnota je menší než 32, tj. všechny znaky, které se v ASCII tabulce nachází před mezerou). Výsledný řetězec vrátí jako svoji funkční hodnotu.

```
Př.:  Trim(#7+' Hokus Pokus '+#10) = 'Hokus Pokus'
```

6.9. TrimLeft

```
function TrimLeft(const S: string): string;
```

Funkce **TrimLeft** odstraní z řetězce S všechny počáteční mezery a řídicí znaky (znaky, jejichž ASCII hodnota je menší než 32, tj. všechny znaky, které se v ASCII tabulce nachází před mezerou). Výsledný řetězec vrátí jako svoji funkční hodnotu.

```
Př.:  TrimLeft(#7+' Hokus Pokus '+#10) = 'Hokus Pokus '#10
```

6.10. TrimRight

```
function TrimRight(const S: string): string;
```

Funkce **TrimRight** odstraní z řetězce S všechny koncové mezery a řídicí znaky (znaky, jejichž ASCII hodnota je menší než 32, tj. všechny znaky, které se v ASCII tabulce nachází před mezerou). Výsledný řetězec vrátí jako svoji funkční hodnotu.

```
Př.:  TrimRight(#7+' Hokus Pokus '+#10) = #7' Hokus Pokus'
```

6.11. Enlarge

```
function Enlarge     (const S: string; Len: byte; Zn: char): string;
```

Funkce **Enlarge** rozšíří řetězec S střídavě zleva i zprava znaky Zn na celkovou délku Len. Tj. řetězec S je ve výsledném řetězci vycentrován na střed. Pokud řetězec S je delší nebo roven Len, rozšíření se neprovede. Výsledný řetězec funkce vrátí jako svoji funkční hodnotu.

```
Př.:  Enlarge('Centered String',20, '-') = '--Centered String---'
```

6.12. EnlargeLeft

```
function EnlargeLeft (const S: string; Len: byte; Zn: char): string;
```

Funkce **EnlargeLeft** rozšíří řetězec S zleva znaky Zn na celkovou délku Len. Tj. řetězec S je ve výsledném řetězci zarovnán doleva. Pokud řetězec S je delší nebo roven Len, rozšíření se neprovede. Výsledný řetězec funkce vrátí jako svoji funkční hodnotu.

```
Př.:  EnlargeLeft('Left String',20, '-') = 'Left String---------'
```

6.13. EnlargeRight

```
function EnlargeRight(const S: string; Len: byte; Zn: char): string;
```

Funkce **EnlargeRight** rozšíří řetězec S zprava znaky Zn na celkovou délku Len. Tj. řetězec S je ve výsledném řetězci zarovnán doprava. Pokud řetězec S je delší nebo roven Len, rozšíření se neprovede. Výsledný řetězec funkce vrátí jako svoji funkční hodnotu.

```
Př.:  EnlargeRight('Right String',20, '-') = '--------Right String'
```

6.14. EquStr

```
function EquStr(var S,D; L:byte): Boolean;
```

Funkce **EquStr** testuje shodnost jakýchkoli dvou datových struktur do délky 255 byte. Parametry S a D určují počáteční adresy struktur, od kterých se bude porovnávat, a parametr L určuje délku porovnávání v počtu byte. Při shodnosti obou struktur vrací funkce hodnotu True.

6.15. StringOfChar

```
function StringOfChar(Chr: char; I: byte): string;
```

Funkce **StringOfChar** vrátí jako svoji funkční hodnotu řetězec, který obsahuje I znaků Chr.

```
Př.:  StringOfChar('A',5) = 'AAAAA'
```

6.16. SetLength

```
procedure SetLength(var S:string; NewLength:byte);
```

Procedura **SetLength** nastaví délku řetězce S na NewLength. V jazyce Borland Pascal 7.0 je délka řetězce uložena v nultém znaku řetězce (S[0]). Ve vyšších jazycích např. Delphi existují i jinak definované řetězce. Proto pro případný budoucí přechod

aplikace např. do Delphi by měl programátor používat výhradně této procedury a ne přímého nastavení nultého (délkového) znaku řetězce.

Pokud aplikace definuje řetězec MyString délky např. 20 (`var MyString : string[20]`) a chce zavolat např. funkci `SetLength(MyString,5)`, nesmí být v překladači zapnuta direktiva $V (Strict Var Strings). Dále aplikace musí zajistit, aby zadávaná délka NewLength nebyla větší než maximální délka deklarovaného řetězce, tj. např. volání `SetLength(MyString,21)` je chybné – funkce tuto kontrolu provést nemůže.

6.17. CompareStr

```
function CompareStr(const S1, S2: string): Integer;
```

Funkce **CompareStr** porovná textové řetězce S1 a S2 podle hodnot jejich ASCII znaků s rozlišováním velkých a malých písmen.

Je-li S1 < S2, vrátí číslo menší než 0.
Je-li S1 = S2, vrátí 0.
Je-li S1 > S2, vrátí číslo vetší než 0.

```
Př.:  CompareStr('Pokus A', 'Pokus B') = -1;
      CompareStr('Pokus A', 'Pokus A') =  0;
      CompareStr('Pokus AA','Pokus A') =  1;
```

6.18. CompareText

```
function CompareText(const S1, S2: string): Integer;
```

Funkce **CompareText** porovná text v řetězcích S1 a S2 bez s rozlišování velkých a malých písmen, tj. ignoruje rozdíly mezi znaky ‘a’ až ‘z’ a ‘A’ až ‘Z’.

Je-li S1 < S2, vrátí číslo menší než 0.
Je-li S1 = S2, vrátí 0.
Je-li S1 > S2, vrátí číslo vetší než 0.

```
Př.:  CompareText('Pokus A', 'Pokus B') = -1;
      CompareText('Pokus A', 'Pokus a') =  0;
```

6.19. PosText

```
function PosText(const SubStr, S: string): Byte;
```

Funkce **PosText** pracuje podobně jako standardní Pascalská funkce Pos (tj. vyhledá podřetězec SubStr v řetězci S) s tím rozdílem, že nerozlišuje velikost písma. Jako funkční hodnotu vrátí index do řetězce S, kde se nalézá začátek podřetězce SubStr. Pokud podřetězec SubStr v řetězci S nenajde, vrátí 0.

```
Př.:  {standardní Pascalská funkce Pos}
       Pos('kus', 'Pokus') = 3;
       Pos('Kus', 'Pokus') = 0;
      {funkce PosText}
       PosText('Kus', 'POKUS') = 3;
       PosText('Kus', 'Pokus') = 3;
```

6.20. PosI

```
function PosI(const SubStr, S: string; I:byte): Byte;
```

Funkce **PosI** pracuje podobně jako standardní Pascalská funkce Pos (tj. vyhledá podřetězec SubStr v řetězci S) s tím rozdílem, že řetězec S prohledává od I-té

pozice včetně. Jako funkční hodnotu vrátí index do řetězce S, kde se nalézá začátek podřetězce SubStr. Pokud podřetězec SubStr v řetězci S od I-té pozice nenajde, vrátí 0.

```
Př.: {standardní Pascalská funkce Pos}
      Pos ('kus', 'Hokus Pokus') = 3;
     {funkce PosI} {12345678901}
      PosI('kus', 'Hokus Pokus', 5) = 9;
      PosI('kus', 'Hokus Pokus', 1) = 3;
      PosI('kus', 'Hokus Pokus',10) = 0;
```

6.21. PosW

```
function PosW(const SubStr,S:string; const TextChars:tCharSet):Byte;
```

Funkce **PosW** pracuje podobně jako standardní Pascalská funkce Pos (tj. vyhledá podřetězec SubStr v řetězci S) s tím rozdílem, že hledaný řetězec nesmí být v řetězci S ohraničen znaky z množiny TextChars, tj. funkce hledá pouze „celá slova" SubStr v řetězci S. Jako funkční hodnotu vrátí index do řetězce S, kde se nalézá začátek podřetězce SubStr. Pokud podřetězec SubStr v řetězci S neohraničený znaky TextChars nenajde, vrátí 0.

```
Př.: {standardní Pascalská funkce Pos}
      Pos ('kus', 'Hokus kus') = 3;
    {funkce PosW} {123456789}
      PosW('kus', 'Hokus kus', ['a'..'z','A'..'Z']) = 7;
```

6.22. ReverseStr

```
function ReverseStr(const S:string):string;
```

Funkce **ReverseStr** přehází znaky v řetězci S a výsledek vrátí jako svoji funkční hodnotu.

```
Př.: ReverseStr('Pokus') = 'sukoP'
```

6.23. WrapText

```
function WrapText(const Line:string; MaxCol:byte):string;
```

Funkce **WrapText** zalomí slova řetězce Line na maximální délku MaxCol jednoho řádku. Jako oddělovače slov používá znaky v množině WordDelimiters. Jako zalomení řádky používá znaky #13#10. Je-li některé slovo delší než MaxCol, rozdělí ho (ale nepoužívá znaku '-' pro dělení slov). Výsledný zalomený řetězec vrátí jako svoji funkční hodnotu.

Je nutno dát pozor, pokud řetězec Line je hodně dlouhý. Při vkládání znaků #13#10 by se mohlo stát, ze výsledný řetězec přeteče velikost 255 znaků. V takovém případě funkce vrátí maximální řetězec, tj. některá slova z konce mohou vypadnout.

```
Př.: WrapText('Toto je pokusny retezec pro funkci WrapText',10) =
     'Toto je '#13#10'pokusny '#13#10'retezec '#13#10'pro
     funkci'#13#10' WrapText'
```

6.24. NDelim

```
function NDelim(const Slovo:WordString; const TABDelim:String):Byte;
```

Funkce **NDelim** vyhledá zadaný řetězec Slovo v tabulce klíčových slov TABDelim a vrátí jeho index v této tabulce. Parametrem TABDelim je řetězec, kde jsou jednotlivá slova za sebou uvedena a jsou oddělena oddělovačem '|'. Délka těchto

slov je dána délkou prvního slova v TABDelim (ostatní slova musí být stejné délky - například je vhodné je doplnit mezerami). Pokud funkce zadané slovo v tabulce TABDelim nenalezne, vrátí hodnotu 0.

```
Př.:  Ndelim('NOD','COM|ADR|NOD|IRQ') = 3
      Ndelim('nod','COM|ADR|NOD|IRQ') = 0
```

6.25. AddParamStr

```
function AddParamStr(BaseParamStr: tParamStr;
                     const NewParamStr : tParamStr): tParamStr;
```

Funkce **AddParamStr** upraví řetězec BaseParamStr, který representuje řetězec parametrů s hodnotami. Úpravu provádí podle řetězce NewParamStr následovně dvěma způsoby:

1. Aktualizují se hodnoty těch parametrů, které jsou obsaženy v řetězci BaseParamStr a zároveň i v řetězci NewParamStr.
2. Z řetězce NewParamStr se přidají nové parametry, které nejsou v řetězci BaseParamStr, i s jejich hodnotami.

Výsledek pak vrátí jako svou funkční hodnotu. Tuto funkci lze použít například pro úpravu inicializačních řetězců komunikačních knihoven.

```
Př.: AddParamStr('NAM=COM COM=1 IRQ=4 PAR=N', 'PAR=O STO=1')
               = 'NAM=COM COM=1 IRQ=4 PAR=O STO=1';
```

6.26. SubParamStr

```
function SubParamStr(BaseParamStr: tParamStr;
                     const NewParamStr : tParamStr): tParamStr;
```

Funkce **SubParamStr** upraví řetězec BaseParamStr, který representuje řetězec parametrů s hodnotami, tak, že z řetězce BaseParamStr odstraní ty parametry i s jejich hodnotami, které jsou obsaženy i v řetězci NewParamStr. Na hodnotě odstraňovaných parametrů v řetězci NewParamStr nezáleží, dokonce nemusí být ani uvedena.
Výsledek pak vrátí jako svou funkční hodnotu. Tuto funkci lze použít například pro úpravu inicializačních řetězců komunikačních knihoven.

```
Př.: SubParamStr('NAM=COM COM=1 IRQ=4 PAR=N', 'PAR COM')
               = 'NAM=COM IRQ=4';
```