Použití systémových funkcí API

v jazyku ILE RPG verze 7. 3

Vladimír Župka

Obsah

Úvod

Kurs je určen programátorům znalým jazyka RPG IV, verze 7.3, kteří chtějí používat funkcí systému IBM i. Tento text vychází ze staršího textu, ale mění ukázky programů z pevného formátu na volný.

Systémové funkce vystupují pod názvem *Application Programming Interface*, zkráceně *API*. Říká se jim také *volání API*, *programy API* apod. Je jich mnoho a jsou v podstatě dvojího druhu.

Starší funkce jsou k dispozici od začátku a lze je používat i v programovém modelu OPM (Original Program Model), tedy v jazyku RPG/400 čili RPG III (nebo CL/400, COBOL/400, C/400).

Novější funkce jsou k dispozici až po zavedení programového modelu ILE (Integrated Language Environment), tedy pouze v jazyku ILE/RPG čili RPG IV (nebo ILE/CL, ILE/COBOL, ILE/C, ILE/C++).

Obecný popis a návod k použití nalezneme zde:

Programming API overview and concepts.

Podrobná dokumentace je zde:

https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/apiref/api.htm.

Na této stránce nalezneme pod hlavičkou "APIs by category" seznam tematických skupin API. Z něj uvádíme jen některé:

Configuration
Database and File
Message Handling
Object
UNIX(R)-Type
Work Management

Ze skupiny *UNIX(R)-Type* uvádíme tyto skupiny API:

Integrated File System APIs
Sockets APIs
Header Files for UNIX-Type Functions

Skupina *UNIX(R)-Type* obsahuje funkce převzaté z operačních systémů typu Unix. Tyto funkce jsou psány v jazyku C a jsou tedy dokumentovány s ohledem na tento jazyk. Vyžadují popis prototypu, který je v případě potřeby nutné ručně převést z jazyka C do jazyka RPG (popř. COBOL nebo CL).

Ostatní skupiny API nepotřebují žádné popisy prototypů, mnohdy však vyžadují poměrně složité datové struktury pro zadání vstupních a výstupních parametrů. Některé funkce API těchto skupin doplňují nebo nahrazují příkazy CL typu RTVxxx, popř. DSPxxx. Ostatní slouží speciálním účelům. Často je lze používat i v jazyku RPG III. Některé jednodušší funkce poskytují jako výsledek *jednotlivou informaci*, složitější poskytují informace v podobě *seznamu údajů*. Jiné funkce umožňují provedení určité činnosti.

Práce s jedním záznamem

Získání informace o úloze

Na příkladu funkce QUSRJOBI (Retrieve Job Information) ukážeme, jak se získá jednotlivý záznam s informacemi o úloze. Tato funkce se dá přirovnat CL příkazu RTVJOBA v tom smyslu, že získává podobné údaje.

Funkce QUSRJOBI je popsána ve skupině *Work Management* pod názvem *Retrieve Job Information*. Popis je velmi dlouhý a členitý. V podstatě jde o popis vstupních (Input), výstupních (Output) a obousměrných (I/O) parametrů volání. Často bývá část parametrů povinná a část (koncová) nepovinná.

Parametry volání

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified job name	Input	Char(26)
5	Internal job identifier	Input	Char(16)

Optional Parameter Group 1:

6	Error code	I/O	Char(*)

Optional Parameter Group 2:

7	Reset performance statistics	Input	Char(1)

1. První povinný parametr představuje výsledek a znamená vlastně datovou strukturu, jejíž délku si určíme sami (hvězdička v závorce znamená neurčenou délku).
2. Délku výsledku zadáváme v druhém parametru podle toho, který formát zvolíme ve třetím parametru.
3. Osmiznakové jméno formátu volíme podle toho, které údaje o úloze chceme získat. Zde jsou k dispozici tyto formáty:

   JOBI0100 Basic performance information
   JOBI0150 Additional performance information
   JOBI0200 WRKACTJOB information
   JOBI0300 Job queue and output queue information
   JOBI0400 Job attribute information
   JOBI0500 Message logging information
   JOBI0600 Active job information
   JOBI0700 Library list information
   JOBI0750 Extended library list information
   JOBI0800 Active job signal information
   JOBI0900 Active job SQL information
   JOBI1000 Elapsed performance statistics

   Například datová struktura formátu JOBI0300 má tento tvar:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(10)	Job queue name
72	48	CHAR(10)	Job queue library name
82	52	CHAR(2)	Job queue priority
84	54	CHAR(10)	Output queue name
94	5E	CHAR(10)	Output queue library name
104	68	CHAR(2)	Output queue priority
106	6A	CHAR(10)	Printer device name
116	74	CHAR(10)	Submitter's job name
126	7E	CHAR(10)	Submitter's user name
136	88	CHAR(6)	Submitter's job number
142	8E	CHAR(10)	Submitter's message queue name
152	98	CHAR(10)	Submitter's message queue library name
162	A2	CHAR(10)	Status of job on the job queue
172	AC	CHAR(8)	Date and time job was put on this job queue
180	B4	CHAR(7)	Job date

4. Kvalifikované jméno úlohy se skládá ze jména úlohy, jména uživatele a čísla úlohy (délky 10, 10, 6).
5. Interní identifikátor úlohy zůstává normálně prázdný, pro speciální požadavky jej lze získat předem jinou systémovou funkcí. Slouží ke zvýšení rychlosti.

Nepovinné parametry lze použít pro speciální účely.

6. Chybový kód je datová struktura, do níž funkce uloží informace o chybě, k níž došlo při volání. Nezadáme-li tento parametr, je chyba hlášena standardně.
7. Tento parametr se týká jen formátu JOBI1000, který získává výkonové statistiky úlohy.

Zápis datových struktur a volání funkce

Systémové funkce používají datové struktury, které určují rozvržení zadávaných i získaných informací. Tyto struktury je výhodné zkopírovat z knihovny *QSYSINC*, z příslušného zdrojového souboru, pro nás je to *QRPGLESRC*. Tam najdeme pro každou funkci zdrojový člen, v němž jsou zapsány definice příslušných datových struktur. Tyto struktury jsou zapsány ve starší verzi jazyka RPGIV v pevném formulářovém formátu. Lze je však do zdrojového členu s volným formátem přikopírovat direktivou /COPY nebo /INCLUDE. Následující datová struktura může být získána jako součást většího počtu struktur pomocí direktivy

/COPY QSYSINC/QRPGLESRC,QUSRJOBI

```
 *   Datová struktura pro API QUSRJOBI
D*************************************************************
D*Record structure for QUSRJOBI JOBI0300 format
D*************************************************************
DQUSI030000       DS
D*                                          Qwc JOBI0300
D QUSBR02                 1      4B 0
```

```
D*                                        Bytes Return
D QUSBA02                 5      8B 0
D*                                        Bytes Avail
D QUSJN04                 9     18
D*                                        Job Name
D QUSUN04                19     28
D*                                        User Name
D QUSJNBR04              29     34
D*                                        Job Number
D QUSIJID02              35     50
D*                                        Int Job ID
D QUSJS07                51     60
D*                                        Job Status
D QUSJT05                61     61

...

D QUSJD                 181    187
D*                                        Job Date

//  Parametry pro API QUSRJOBI
Dcl-S del_bin               Int(10:0) inz(%size(QUSI030000));
Dcl-S format               Char(8) inz('JOBI0300');
Dcl-S jobname              Char(26) inz('*');
Dcl-S int_job_id           Char(16) inz(' ');

//  Prototyp API QUSRJOBI
Dcl-PR  usrjobi extpgm('QUSRJOBI'); // Program se nemůže jmenovat stejně jako API,
                       // protože tak je pojmenovaná konstanta v datové struktuře.
   QUSI030000            LikeDS(QUSI030000); // Odvolává se na předchozí strukturu
   del_bin                  Int(10:0);
   format                  Char(8);
   jobname                 Char(26);
   int_job_id              Char(16);
End-PR  usrjobi;

//  Volání API QUSRJOBI pro získání údajů o úloze
//     Výsledek se uloží do datové struktury QUSI030000
callp usrjobi ( QUSI030000:    // Výsledek
                del_bin:       // Délka výsledku
                format:        // Jménoformátu
                jobname:       // Jméno úlohy = *
                int_job_id );  // Interní ID = ' '
```

Parametr *jobname* (jméno úlohy) jsme zadali jako hvězdičku, což znamená úlohu, v níž právě běží tento program. Jinak bychom museli zadat celé kvalifikované jméno úlohy, které bychom museli znát odjinud, třeba zjistit jinou funkcí API, např. List Job (QUSLJOB).

Místo přikopírování datové struktury příkazem /COPY je ovšem možné do programu zapsat vlastní definici datové struktury, například

```
// Datová struktura pro výsledky volání API QUSRJOBI
Dcl-DS  QUSI030000;
   QUSJN04                    Char(10)   Pos(9);   // Job Name
   QUSUN04                    Char(10)   Pos(19);  // User Name
   QUSJNBR04                  Char(06)   Pos(29);  // Job Number
End-DS;
```

Tím se program trochu znázorní.

Příklad

Funkční příklad představuje program KLIENTR (klient), který funguje spolu s programem SERVERR (server). Oba programy jsou uvedeny *v* dodatku o datových frontách, které představují systémové objekty vyžadující k obsluze příslušné funkce API.

Server spuštěný v dávkové nebo interakční úloze čeká na požadavky od klienta spuštěného v interakční úloze. Uživatel klienta zadá z obrazovky číslo zboží pošle je přes datovou frontu (FIFO) serveru, který ji z fronty vyzvedne, najde k tomuto zboží podrobné údaje v souboru ZBOZI a pošle je zpět klientovi přes jinou datovou frontu (klíčovanou).

Součástí zprávy, kterou klient posílá do datové fronty, je jeho identifikace. Je dlouhá 36 bajtů a skládá se ze jména úlohy, jména uživatele a čísla úlohy. Tyto údaje si klient získá právě voláním funkce QUSRJOBI.

Když pak server posílá odpověď klientovi přes klíčovanou frontu, použije jeho identifikaci jako klíč zprávy. Klient si pak vybere správnou odpověď podle 36bajtového klíče rovného své identifikaci. Klientů proto může být spuštěno více a odpovědi od serveru si nepopletou.

Posílání a zachycování zpráv

Zde použijeme funkci QMHRCVPM (Receive Program Messages), která funguje podobně jako CL příkaz RCVMSG pro programové zprávy.

Parametry volání

Required Parameter Group:

1	Message information	Output	Char(*)
2	Length of message information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Call stack entry	Input	Char(*) or Pointer
5	Call stack counter	Input	Binary(4)
6	Message type	Input	Char(10)
7	Message key	Input	Char(4)
8	Wait time	Input	Binary(4)
9	Message action	Input	Char(10)
10	Error code	I/O	Char(*)

Optional Parameter Group 1:

11	Length of call stack entry	Input	Binary(4)
12	Call stack entry qualification	Input	Char(20)

Optional Parameter Group 2:

13	Call stack entry data type	Input	Char(10)
14	Coded character set identifier	Input	Binary(4)

Optional Parameter Group 3:

15	Allow default reply rejection	Input	Char(10)

1. První povinný parametr představuje výsledek (hvězdička v závorce znamená neurčenou délku) tedy vlastně datovou strukturu, jejíž délku si určíme sami (bod 2) a jejíž tvar určíme volbou jména formátu (bod 3).
2. Délku výsledku zadáváme v druhém parametru podle toho, který formát zvolíme ve třetím parametru.
3. Osmiznakové jméno formátu volíme podle toho, které údaje o úloze chceme získat. Zde jsou k dispozici tyto formáty:

   RCVM0100 Brief message information.
   RCVM0200 All message information.
   RCVM0300 All message information. Complete sender information for the message being received.
4. Položka zásobníku volání. Z několika možností volíme zápis hvězdičky, která znamená současnou položku, tedy tu, která patří současné proceduře (v níž voláme tuto funkci API).
5. Čítač položek zásobníku. Lze zadat relativní pozici, vzhledem položce zadané v předchozím parametru. Nula znamená tuto položku, kladné číslo *n* znamená *n*-tou předchozí položku.
6. Typ zprávy. Lze zadat hodnoty: *FIRST, *LAST, *PRV, *NEXT, *ESCAPE, *NOTIFY a další. U mnoha typů se nemusí zadat klíč zprávy, dokonce u typů *FIRST a *LAST není ani dovolen.

7. Klíč zprávy. Mezery znamenají nezadaný klíč. Klíč můžeme zadat jako binární čtyřbajtový údaj, který lze zjistit předem voláním této funkce bez klíče. Pro různé kombinace typu a klíče zprávy platí jistá pravidla. Zadáme-li klíč, funkce vybere z fronty existující zprávu s tímto klíčem, jinak hlásí chybu.
8. Čekací doba. Číslo 0 znamená nečekat na zprávu. Číslo -1 znamená čekat, dokud zpráva nepřijde do fronty. Číslo větší než 0 znamená počet sekund čekání na zprávu.
9. Akce po získání zprávy. Kód *OLD nechává zprávu ve frontě označenou jako starou. Kód *REMOVE odstraní zprávu z fronty. Kód *SAME nechá zprávu ve frontě v původním stavu.
10. Chybový kód je datová struktura, do které funkce zaznamená chybovou zprávu o svém průběhu.

Offset		**Use**	**Type**	**Field**
Dec	**Hex**			
0	0	INPUT	BINARY(4)	Bytes provided
4	4	OUTPUT	BINARY(4)	Bytes available
8	8	OUTPUT	CHAR(7)	Exception ID
15	F	OUTPUT	CHAR(1)	Reserved
16	10	OUTPUT	CHAR(*)	Exception data

První parametr *Bytes provided* zadává počet bajtů, který je k dispozici pro funkci. Nula znamená, že chyba bude hlášena standardním způsobem do fronty zpráv. Číslo 8 nebo větší znamená, že funkce zapíše informaci o chybě do dalších parametrů (Exception ID a Exception data).

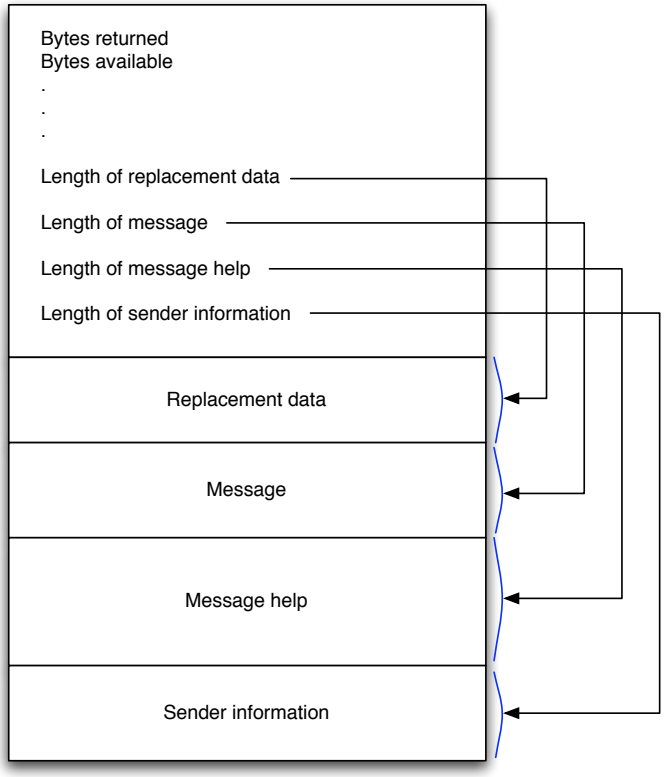
Druhý parametr *Bytes available* zadává počet bajtů informace, kolik funkce vrátila.

Třetí parametr *Exception ID* je sedmimístné číslo chybové zprávy,

Pátý parametr *Exception data* je text chybové zprávy.

Nepovinnými parametry volání se nebudeme zabývat, mají okrajový význam.

V příkladu použijeme formát RCVM0300, který lze znázornit takto:



Příklad

Fungující příklad představuje program RCVMSG. Ten záměrně produkuje chyby, které monitoruje a získává z nich chybové zprávy pomocí funkce QMHRCVPM. První chyba je programová - *dělení nulou*, druhá chyba je souborová - *přepis záznamu* (UPDATE) *bez jeho předchozího přečtení*.

Výsledky jsou vidět v několika výpisech paměti. První výpis se týká chyby dělení nulou. Další výpisy odrážejí všechny zprávy chyby přepisu záznamu - od poslední do první zprávy programové fronty zpráv ze současné položky zásobníku volání.

Formát RCVM0300 pro výsledky přijetí zprávy z programové fronty

Datová struktura tohoto formátu je obsažena v souboru QSYSINC/QRPGLESRC(QMHRCVPM) společně s dalšími formáty. Zde je uvedena jako zdroj informací pro definici datové struktury v programu ve stručnější verzi.

```
 *----------------------------------------------------------------
 *   Formát RCVM0300 pro výsledky přijetí zprávy z programové fronty
 *   QSYSINC/QRPGLESRC,QMHRCVPM - lehce upraven
 *----------------------------------------------------------------
DQMHM0300         DS
D*                                        Qmh Rcvpm RCVM0300
D QMHBRTN03               1      4B 0
D*                                        Bytes Returned
D QMHBAVL04               5      8B 0
D*                                        Bytes Available
D QMHMS07                 9     12B 0
D*                                        Message Severity
D QMHMI05                13     19
D*                                        Message Id
D QMHMT07                20     21
D*                                        Message Type
D QMHMK05                22     25
D*                                        Message Key
D QMHMFILN03             26     35
```

```
D*                                              Message File Name
D QMHMFILL01              36     45
D*                                              Message File Library
D QMHMLIBU01              46     55
D*                                              Message Library Used
D QMHAO01                 56     64
D*                                              Alert Option
D QMHSIDCS05              65     68B 0
D*                                              Text CCSID Convert Status
D QMHSIDCS06              69     72B 0
D*                                              Data CCSID Convert Status
D QMHCSIDR05              73     76B 0
D*                                              Text CCSID Returned
D QMHCSIDR06              77     80B 0
D*                                              Data CCSID Returned
D QMHLDRTN01              81     84B 0
D*                                              Length Data Returned
D QMHLDAVL01              85     88B 0
D*                                              Length Data Available
D QMHLMRTN01              89     92B 0
D*                                              Length Message Returned
D QMHLMAVL01              93     96B 0
D*                                              Length Message Available
D QMHLHRTN01              97    100B 0
D*                                              Length Help Returned
D QMHLHAVL01             101    104B 0
D*                                              Length Help Available
D QMHLSRTN               105    108B 0
D*                                              Length Send Returned
D QMHLSAVL               109    112B 0
D*                                              Length Send Available
D msg_data_beg           113    800
```

Text programu RCVMSG

```
**free
//*****************************************************************
//   RCVMSG - Získá údaje o zprávě z programové fronty
//*****************************************************************
//=================================================================
//   Popis souborů
//=================================================================
Dcl-F CENY                   Usage(*UPDATE:*DELETE);
//=================================================================
//   Definice dat
//=================================================================

//-----------------------------------------------------------------
//   Formát RCVM0300 pro výsledky přijetí zprávy z programové fronty
//   QSYSINC/QRPGLESRC,QMHRCVPM) - zkrácen
//-----------------------------------------------------------------
Dcl-DS QMHM0300;
   QMHBRTN03              BinDec(8:0) Pos(1);   // Bytes Returned
   QMHBAVL04              BinDec(8:0) Pos(5);   // Bytes Available
   QMHMK05                  Char(4) Pos(22);    // Message Key
   QMHLDRTN01             BinDec(8:0) Pos(81);  // Length Data Returned
   QMHLMRTN01             BinDec(8:0) Pos(89);  // Length Message Returned
   QMHLHRTN01             BinDec(8:0) Pos(97);  // Length Help Returned
   msg_data_beg             Char(697) Pos(113);
End-DS;

//-----------------------------------------------------------------
//   Datová struktura chybového kódu podle členu
//   QSYSINC/QRPGLESRC,QUSEC
//-----------------------------------------------------------------
Dcl-DS QUSEC;
```

```
   QUSBPRV                BinDec(8:0) Pos(1); //  Bytes Provided
   QUSBAVL                BinDec(8:0) Pos(5); //  Bytes Available
   QUSEI                    Char(7) Pos(9);   //  Exception Id
   QUSERVED                 Char(1) Pos(16);  //  Reserved
   err_msg_beg              Char(71) Pos(17);
End-DS;

//   Parametry pro API QMHRCVPM
Dcl-S msg_info_len           Int(10:0) inz(%len(QMHM0300)); // viz předchozí
                                                            // strukturu
Dcl-S format               Char(8) inz('RCVM0300');
Dcl-S callstack_ent        Char(10) inz('*');
Dcl-S callstack_cnt         Int(10:0) inz(0);
Dcl-S msg_type             Char(10) inz('*ANY');
Dcl-S msg_key              Char(4) inz(*blank);
Dcl-S wait_time             Int(10:0) inz(5);
Dcl-S msg_action           Char(10) inz('*SAME');
Dcl-S calstkent_len         Int(10:0) inz(10);
Dcl-S calstkent_qual       Char(20) inz('*NONE     *NONE     ');
Dcl-S calstkent_typ        Char(10) inz('*CHAR');
Dcl-S CCSID                 Int(10:0) inz(0);
Dcl-S alwdftrpl_rej        Char(10) inz('*NO');

//   Pracovní proměnné
Dcl-S msg_data             Char(200);
Dcl-S msg                  Char(200);
Dcl-S msg_help             Char(500);

Dcl-S delenec             Packed(5:0) inz(10);
Dcl-S delitel             Packed(5:0);
Dcl-S podil               Packed(5:0);
Dcl-S msg_info_off           Int(10:0);
Dcl-S dump_label           Char(10) inz('Ladění');
Dcl-S idx                   Int(10:0);

// Prototyp API QMHRCVPM
Dcl-PR QMHRCVPM extpgm('QMHRCVPM');
   QMHM0300              LikeDS(QMHM0300);
   msg_info_len              Int(10:0);
   format                   Char(8);
   callstack_ent            Char(10);
   callstack_cnt             Int(10:0);
   msg_type                 Char(10);
   msg_key                  Char(4);
   wait_time                 Int(10:0);
   msg_action               Char(10);
   QUSEC                 LikeDS(QUSEC);
   // calstkent_len          Int(10:0);
   // calstkent_qual        Char(20);
   // calstkent_typ         Char(10);
   // CCSID                  Int(10:0);
   // alwdftrpl_rej );      Char(10);
End-PR QMHRCVPM;

//=================================================================
//   Hlavní program
//=================================================================

Monitor;

   delitel = 0;
   podil = delenec / delitel;

On-Error *program;
   msg_type = '*LAST';
   Exsr chyba;
```

```
   idx += 1;
   dump_label = 'Ladění' + %char(idx);
   Dump(A) dump_label;
EndMon;

Monitor;

   Update cenyr;

On-Error *file;
   msg_type = '*LAST';
   msg = *allx'00';
   msg_data = *allx'00';
   msg_help = *allx'00';
   Exsr chyba;
   idx += 1;
   dump_label = 'Ladění' + %char(idx);
   Dump(A) dump_label;
   DoW not (QMHBAVL04 = 0 and QMHBRTN03 = 8);
      msg_key = QMHMK05;
      msg_type = '*PRV';
      msg = *allx'00';
      msg_data = *allx'00';
      msg_help = *allx'00';
      Exsr chyba;
      idx += 1;
      dump_label = 'Ladění' + %char(idx);
      Dump(A) dump_label;
   EndDo;

EndMon;

*inlr = *on;

//====================================================================
//   Podprogramy
//====================================================================
//--------------------------------------------------------------------
//   Chyba - Ošetření chyby pomocí API QMHRCVPM (Receive
//           Program Message)
//--------------------------------------------------------------------
BegSr chyba;
   QUSBPRV = 50;
   QUSBAVL = 0;
   callp QMHRCVPM ( QMHM0300:
                    msg_info_len:
                    format:
                    callstack_ent:
                    callstack_cnt:
                    msg_type:
                    msg_key:
                    wait_time:
                    msg_action:
                    QUSEC         );
                    // calstkent_len:
                    // calstkent_qual:
                    // calstkent_typ:
                    // CCSID:
                    // alwdftrpl_rej

   msg_info_off = %addr(msg_data_beg) - %addr(QMHM0300) + 1;
   If QMHLDRTN01 <> 0;
      msg_data = %subst(QMHM0300 : msg_info_off
           + QMHLDRTN01);
   EndIf;
```

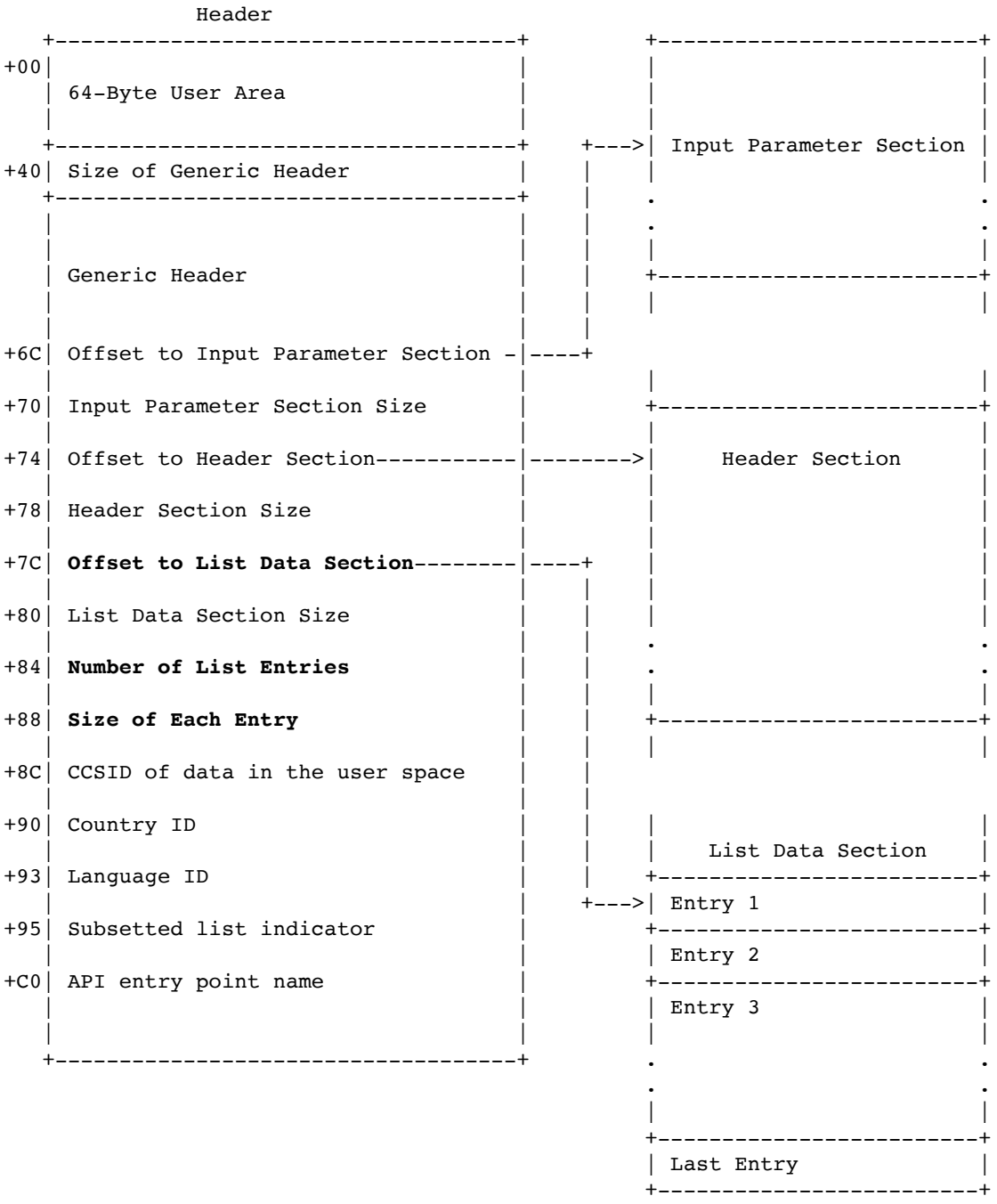
```
   If QMHLMRTN01 <> 0;
      msg  = %subst(QMHM0300 : msg_info_off
           + QMHLDRTN01 : QMHLMRTN01);
   EndIf;

   If QMHLHRTN01 <> 0;
      msg_help = %subst(QMHM0300 : msg_info_off
           + QMHLDRTN01 + QMHLMRTN01 : QMHLHRTN01);
   EndIf;

EndSr;
```

Práce s předem neznámým počtem záznamů

Uživatelský prostor – User Space

Systémové funkce, které jako výsledek poskytují předem neznámý počet záznamů, musí mít k dispozici potřebnou paměť pro výstupní informace. K tomu slouží zvláštní druh objektu, a sice *uživatelský prostor* (*user space*), jehož typ je ***USRSPC**. K jeho vytvoření slouží systémová funkce ***QUSCRTUS*** (*Create User Space*). Prostor je naplněn za použití jiné systémové funkce, zde **QDCLCFGD** (List Configuration Descriptions), jejímž výsledkem je seznam konfiguračních popisů, konkrétně popisů zařízení. Jakmile je takto naplněn uživatelský prostor, můžeme výsledné údaje získat prostřednictvím systémové funkce ***QUSRTVUS*** (*Retrieve User Space*). Uživatelský prostor obsahuje na začátku záhlaví a za ním další položky. Podrobný popis je uveden v podkapitole User spaces and receiver variables kapitoly API concepts.

Tam také nalezneme obrázek znázorňující polograficky strukturu uživatelského prostoru:



Relativní adresy (se znaménkem +) ve schematu jsou uvedeny v hexadecimálním zakódování. Tyto adresy a hodnoty *"offset"* jsou zadány od začátku uživatelského prostoru a počítají se *od nuly*. Pozici pro příkazy RPG získáme přičtením jedničky.

Popis této struktury v jazyku RPG IV je ve členu QSYSINC/QRPGLESRC(QUSGEN). Zde je uveden pro informaci o používaných podpolích.

```
 *----------------------------------------------------------------
 *   Obecné struktury pro API uživatelského prostoru (User Space)
 *   QSYSINC/QRPGLESRC(QUSGEN)
 *----------------------------------------------------------------
D*****************************************************************
D*Type Definition for the User Space Generic Header.
D*****************************************************************
DQUSH0100         DS
D*                                          Qus Generic Header 0100
D QUSUA                   1     64
D*                                          User Area
D QUSSGH                 65     68B 0
D*                                          Size Generic Header
D QUSSRL                 69     72
D*                                          Structure Release Level
D QUSFN                  73     80
D*                                          Format Name
D QUSAU                  81     90
D*                                          Api Used
D QUSDTC                 91    103
D*                                          Date Time Created
D QUSIS                 104    104
D*                                          Information Status
D QUSSUS                105    108B 0
D*                                          Size User Space
D QUSOIP                109    112B 0
D*                                          Offset Input Parameter
D QUSSIP                113    116B 0
D*                                          Size Input Parameter
D QUSOHS                117    120B 0
D*                                          Offset Header Section
D QUSSHS                121    124B 0
D*                                          Size Header Section
D QUSOLD                125    128B 0
D*                                          Offset List Data
D QUSSLD                129    132B 0
D*                                          Size List Data
D QUSNBRLE              133    136B 0
D*                                          Number List Entries
D QUSSEE                137    140B 0
D*                                          Size Each Entry
D QUSSIDLE              141    144B 0
D*                                          CCSID List Ent
D QUSCID                145    146
D*                                          Country ID
D QUSLID                147    149
D*                                          Language ID
D QUSSLI                150    150
D*                                          Subset List Indicator
D QUSERVED00            151    192
D*                                          Reserved
```

Získání seznamu zařízení

Program DEVICES používá systémovou funkci – API **QDCLCFGD**, která vytvoří seznam zvolených popisů zařízení do uživatelského prostoru. Z tohoto prostoru pak údaje tiskne do souboru QPRINT. Popis funkce QDCLCFGD je uveden v tematické skupině nazvané *Configuration*.

Datová struktura formátu CFGD0200 potřebná k získání výsledků z API QDCLCFGD pro získání konfiguračních údajů o zařízení.

```
*****************************************************************
 *   Type Definition for the CFGD0200 Format.
 *****************************************************************
DQDCD0200         DS
D*                                              Qdc CFGD0200
D QDCCSN                  1      4B 0
D*                                              Current Status Numeric
D QDCCN00                 5     14
D*                                              Config Name
D QDCCDC00               15     24
D*                                              Config Desc Cat
D QDCCST                 25     44
D*                                              Current Status Text
D QDCTD                  45     94
D*                                              Text Desc
D QDCJN                  95    104
D*                                              Job Name
D QDCUN                 105    114
D*                                              User Name
D QDCJNBR               115    120
D*                                              Job Number
D QDCPTD                121    130
D*                                              Pass Thru Device
```

Zdrojový text programu DEVICES

```
//*****************************************************************
//   Program DEVICES - Získání popisů zařízení
//*****************************************************************

//=================================================================
//   Popis souborů
//=================================================================
Dcl-F QPRINT  Printer(300) OflInd(*InOA);
//=================================================================
//   Definice dat
//=================================================================
//   Parametry pro funkci QUSCRTUS (vytvoření uživatelského prostoru)
Dcl-S space_name            Char(20) inz('CFGLST    QTEMP     ');
Dcl-S ext_attrib            Char(10) inz('QDCLCFGD');
Dcl-S space_size             Int(10:0) inz(20000);
Dcl-S space_init            Char(1) inz(x'00');
Dcl-S space_auth            Char(10) inz('*ALL');
Dcl-S space_text            Char(50) inz(*blank);
Dcl-S space_replace         Char(10) inz('*YES');
Dcl-S space_domain          Char(10) inz('*USER');

//   Parametry pro funkci QDCLCFGD (seznam zařízení)
Dcl-S format                Char(8) inz('CFGD0200');
Dcl-S cfg_type              Char(10) inz('*DEVD');
Dcl-DS obj_qualif;  // *N je nutné pro bezejmenná podpole
       *N                    Char(10) inz('*ALL');
       *N                    Char(10) inz;
       *N                    Char(10) inz;
       *N                    Char(10) inz;
End-DS;
Dcl-DS status_qualif         Len(20);
       *N                    Char(10) inz('*EQ');
```

```
       *N                    Char(10) inz('*ACTIVE');
End-DS;

// Prototyp API QUSCRTUS
Dcl-PR QUSCRTUS extpgm ('QUSCRTUS');
   space_name               Char(20)  ;
   ext_attrib               Char(10)  ;
   space_size                Int(10:0);
   space_init               Char(1)   ;
   space_auth               Char(10)  ;
   space_text               Char(50)  ;
   space_replace            Char(10)  ;
   QUSEC                  LikeDS(QUSEC);
   space_domain             Char(10)  ;
End-Pr QUSCRTUS;

// Prototyp API QDCLCFGD
Dcl-PR DCLCFGD extpgm ('QDCLCFGD');
   space_name               Char(20)  ;
   format                   Char(8)   ;
   cfg_type                 Char(10);
   obj_qualif             LikeDS(obj_qualif);
   status_qualif          LikeDS(status_qualif);
   QUSEC                  LikeDS(QUSEC);
End-Pr DCLCFGD;

// Prototyp API QUSRTVUS
Dcl-PR QUSRTVUS extpgm ('QUSRTVUS');
   space_name               Char(20)  ;
   start_pos                 Int(10:0);
   QUSSGH                   Like(QUSSGH);
   QUSH0100                 Like(QUSH0100); // Ne QUSH0100 - je delší!
   QUSEC                    LikeDS(QUSEC);
End-Pr QUSRTVUS;

// Jiný prototyp API QUSRTVUS
Dcl-PR USRTVUS extpgm ('QUSRTVUS');
   space_name               Char(20)  ;
   start_pos                 Int(10:0);
   QUSSGH                   Like(QUSSGH);
   QDCD0200                 Like(QDCD0200);
   QUSEC                    LikeDS(QUSEC);
End-Pr USRTVUS;

//   Pracovní proměnné
Dcl-S start_pos              Int(10:0);
Dcl-S index                  Int(10:0);  // Doplněno
Dcl-S gen_hdr_len            Int(10:0) inz(%size(QUSH0100));

//-----------------------------------------------------------------
//   Záhlaví uživatelského prostoru (User Space)
//   QSYSINC/QRPGLESRC(QUSGEN)
//-----------------------------------------------------------------
/copy QSYSINC/QRPGLESRC,QUSGEN
//-----------------------------------------------------------------
//   Struktura pro seznam konfiguračních popisů (QDCLCFGD)
//   QSYSINC/QRPGLESRC(QDCLCFGD):
//-----------------------------------------------------------------
/copy QSYSINC/QRPGLESRC,QDCLCFGD
//-----------------------------------------------------------------
//   Struktura pro chybový kód (QUSEC)
//-----------------------------------------------------------------
/copy QSYSINC/QRPGLESRC,QUSEC

//=================================================================
//   Hlavní program
//=================================================================
//   Vytvořit uživatelský prostor (User Space)
callp QUSCRTUS ( space_name:       //  Jmeno User Space
                 ext_attrib:       //  Externí atribut
                 space_size:       //  Velikost User Space
                 space_init:       //  poč.hodnot
```

```
                  space_auth:       //  autorizace
                  space_text:
                  space_replace:
                  QUSEC:            //  error_code
                  space_domain  );

  //   Startovní pozice je 1 (ukazuje na data z API)
  start_pos = 1;

  //   Získat seznam zařízení do User Space.
  //   Program se nesmí jmenovat stejně jako API, protože datová struktura obsahuje
  //   konstantu stejného jména.
  callp DCLCFGD  ( space_name   :
                   format       :    // formát výstupu
                   cfg_type     :    // typ konf.objektu
                   obj_qualif   :    // *ALL / *DSP / ...
                   status_qualif:    // např. *EQ
                   QUSEC         );  // error code

  //   Získat záhlaví uživatelského prostoru (User Space Header)
  QUSSGH = gen_hdr_len; // délka záhlaví
  callp QUSRTVUS ( space_name:
                   start_pos :       // startovní pozice
                   QUSSGH    :       // délka záhlaví
                   QUSH0100  :       // záhlaví
                   QUSEC      );     // error code

  //   Nastavit startovní pozici první položky seznamu získanou ze záhlaví (+1)
  start_pos = QUSOLD + 1;
  //   Cyklus přes všechna zařízení získaná z API. Jejich počet
  //     je QUSNBRLE (proměnná v záhlaví uživ. prostoru).
  For index = 1 To QUSNBRLE;
     //   .Získat další položku z uživatelského prostoru (User Space)
     callp USRTVUS  ( space_name:  // Jiné jméno programu pro API QUSRTVUS
                      start_pos :    // startovní pozice
                      QUSSEE    :    // délka položky
                      QDCD0200  :    // položka (data o konfiguraci)
                      QUSEC      );  // error code

     //   Vytisknout položku
     Except output;

     //   Přičíst délku položky k startovní pozici a získat tak
     //     další položku z uživ. prostoru.
     start_pos += QUSSEE;
  EndFor; // (QUSNBRLE)

  //   Ukončit program
  *InLR = *on;
  Return;

  //================================================================
  //   Popis výstupu
  //================================================================
oQPRINT    e            output         1
  //   Status - číselně
o                       QDCCSN        3
  //   Jméno konfiguračního popisu
o                       QDCCN00            +2
  //   Kategorie konfiguračního popisu
o                       QDCCDC00           +2
  //   Status - text
o                       QDCCST             +2
  //   Popis
o                       QDCTD              +2
```

Zjištění uživatelských profilů

Program PROFILES používá systémové volání API **QSYLAUTU**, které vytváří seznam uživatelských profilů v uživatelském prostoru (user space). Uživatelský prostor si předem vytvoří pomocí API ***QUSCRTUS***. Odtud pak údaje tiskne do souboru QPRINT. Ke čtení položek výsledku tentokrát použije ukazatel (space pointer), který získá pomocí API ***QUSPTRUS***. Datové struktury jsou označeny slovem BASED, protože jsou adresovány ukazatelem.

Zdrojový text programu PROFILES

```
// *******************************************************
// Program PROFILES - získání seznamu uživatelských profilů
// *******************************************************

// Popis tiskového souboru
Dcl-F QPRINT  Printer(132) OflInd(*InOA);

//   Parameters for QUSCRTUS API (user space creation)
Dcl-DS spaceName;
   spcName                 Char(10) inz('PRFLIST');
   spcLib                  Char(10) inz('QTEMP');
End-DS;
Dcl-S spaceSize             Int(10:0) inz(100);
Dcl-S spaceInit             Char(1) inz(X'00');
Dcl-S extAttr               Char(10) inz('QSYLAUTU');
Dcl-S spaceAut              Char(10) inz('*ALL'    );
Dcl-S spaceTxt              Char(50) inz(*BLANKS   );
Dcl-S spaceRepl             Char(10) inz('*YES'    );
Dcl-S spaceDomain           Char(10) inz('*USER'   );

//   Parameters for QSYLAUTU API (List Authorized Users)
Dcl-S format                Char(8) inz('AUTU0200');

//   Pointers to User space and the User profile list
Dcl-S spacePtr           Pointer;
Dcl-S listPtr            Pointer;
Dcl-S index                 Int(10:0);

// Prototyp API QUSCRTUS
Dcl-PR QUSCRTUS extpgm ('QUSCRTUS');
   spaceName              LikeDs(spaceName);
   extAttr                  Char(10)  ;
   spaceSize                 Int(10:0);
   spaceInit                Char(1)   ;
   spaceAut                 Char(10)  ;
   spaceTxt                 Char(50)  ;
   spaceRepl                Char(10)  ;
   QUSEC                   LikeDS(QUSEC);
   spaceDomain              Char(10)  ;
End-Pr QUSCRTUS;

// Prototyp API QUSPTRUS
Dcl-PR QUSPTRUS extpgm ('QUSPTRUS');
   spaceName              LikeDs(spaceName);
   spacePtr             Pointer;
End-PR QUSPTRUS;

// Prototyp API QSYLAUTU
Dcl-PR SYLAUTU extpgm ('QSYLAUTU');
   spaceName              LikeDs(spaceName);
   format                   Char(8)   ;
   QUSEC                   LikeDS(QUSEC);
End-Pr SYLAUTU;
```

```
//-----------------------------------------------------------------
//   User Space Generic Header - based by spacePtr pointer
//   QSYSINC/QRPGLESRC,QUSGEN
//-----------------------------------------------------------------
Dcl-DS QUSH0100                 based(spacePtr); // Qus Generic Header 0100
   QUSUA                  Char(64) Pos(1);     // User Area
   QUSSGH                BinDec(8:0) Pos(65);   // Size Generic Header
   QUSSRL                   Char(4) Pos(69);     // Structure Release Level
   QUSFN                    Char(8) Pos(73);     //  Format Name
   QUSAU                   Char(10) Pos(81);    // Api Used
   QUSDTC                  Char(13) Pos(91);    // Date Time Created
   QUSIS                    Char(1) Pos(104);    // Information Status
   QUSSUS                BinDec(8:0) Pos(105);  // Size User Space
   QUSOIP                BinDec(8:0) Pos(109);  // Offset Input Parameter
   QUSSIP                BinDec(8:0) Pos(113);  // Size Input Parameter
   QUSOHS                BinDec(8:0) Pos(117);  // Offset Header Section
   QUSSHS                BinDec(8:0) Pos(121);  // Size Header Section
   QUSOLD                BinDec(8:0) Pos(125);  // Offset List Data
   QUSSLD                BinDec(8:0) Pos(129);  // Size List Data
   QUSNBRLE              BinDec(8:0) Pos(133);  // Number List Entries
   QUSSEE                BinDec(8:0) Pos(137);  // Size Each Entry
   QUSSIDLE              BinDec(8:0) Pos(141);  // CCSID List Ent
   QUSCID                   Char(2) Pos(145);    // Country ID
   QUSLID                   Char(3) Pos(147);    // Language ID
   QUSSLI                   Char(1) Pos(150);    // Subset List Indicator
   QUSERVED00               Char(42) Pos(151);   // Reserved
End-DS;
//-----------------------------------------------------------------
//   Record structure for AUTU0200 format
//   QSYSINC/QRPGLESRC,QSYLAUTU
//-----------------------------------------------------------------
Dcl-DS QSY0200L00               based(listPtr); // Qsy AUTU0200 List
   QSYUP00                  Char(10) Pos(1);    // User Profile
   QSYGP00                  Char(10) Pos(11);   // Group Profile
   QSYPT                    Char(50) Pos(21);   // Profile Text
   QSYERVED03               Char(2) Pos(71);    // Reserved
   QSYSGNBR00            BinDec(8:0) Pos(73);  // Supp Group Number
   QSYSGN00                 Char(10) Pos(77) DIM(15); // Supp Group Names
   QSYUSRGRP3               Char(1) Pos(227);   //  User Group Ind
   QSYGRPMEM3               Char(1) Pos(228);   // Group Members Ind
End-DS;

//-----------------------------------------------------------------
//   Record structure for Error Code Parameter QUSEC
//-----------------------------------------------------------------
/copy QSYSINC/QRPGLESRC,QUSEC

//=================================================================
//   Hlavní program
//=================================================================
//   Inicializace (získat User Space a první Profile List)
Exsr init;
//   Získat seznam autorizovaných profilů (QSYLAUTU API)
Exsr getList;
//   Opakovat dokud je Information Status = C (complete)
//   (aspoň jednou)
DoU QUSIS = 'C';
   //   .Je-li Status = C nebo P (Partial) a počet položek je > 0 -
   //      - Zpracovat položky seznamu
   If QUSIS = 'C' or QUSIS = 'P' and QUSNBRLE > 0;
      //   ..Nastavit ukazatel do seznamu na první položku
      listPtr = spacePtr + QUSOLD;
      //   ..Zpracovat stanovený počet položek
      For index = 1 To QUSNBRLE;
         //   ...Vytisknout položku
```

```
            Except output;
            //   ...Zvýšit ukazatel o velikost položky
            listPtr += QUSSEE;
         EndFor;
         //   ..Je-li status = P (partial list) - Získat další část seznamu
         If QUSIS = 'P';
            Exsr getList;
         EndIf;

      EndIf;
   EndDo;
   //   End program
   Exsr end;

   //================================================================
   //   Podprogramy
   //================================================================
   //----------------------------------------------------------------
   //   init - Jednorázová inicializace
   //                      - Získání User Space a Profile List
   //----------------------------------------------------------------
   BegSr init;
      //   Vytvořit User Space v QTEMP
      callp QUSCRTUS ( spaceName:        //  Jméno User Space
                       extAttr:          //  Externí atribut
                       spaceSize:        //  Velikost User Space
                       spaceInit:        //  poč.hodnota
                       spaceAut:         //  autorizace
                       spaceTxt:         //  space text
                       spaceRepl:        //  replace
                       QUSEC:            //  error_code
                       spaceDomain  );
      //   Získat ukazatel spacePtr do User Space
      callp QUSPTRUS ( spaceName:
                       spacePtr   );      // ukazatel do User Space
   EndSr;

   //----------------------------------------------------------------
   //   getList - Získat seznam autorizovaných profilů
   //----------------------------------------------------------------
   BegSr getList;
      // Program má jiné jméno než API QSYLAUTU kvůli konstantě stejného jména
      callp SYLAUTU  ( spaceName:        //  Jméno User Space
                       format:           //  Jméno formátu AUTU0200
                       QUSEC     );      //  error_code
   EndSr;
   //----------------------------------------------------------------
   //   end - End routine
   //----------------------------------------------------------------
   BegSr end;
      *inlr = *on;
      Return;
   EndSr;

   //================================================================
   //   Popis výstupu
   //================================================================
oQPRINT    e            output         1
   //   User Profile
o                       QSYUP00
   //   Group Profile
o                       QSYGP00             +2
   //   Profile Text
o                       QSYPT               +2
```

Seznam modulů a procedur

Program DSPMODL používá program API **QBNLMODI**, který dovoluje získat následující informace o modulech do uživatelského prostoru:

- seznam symbolů exportovaných do jiných modulů - formát MODL0100 (*EXPORT),
- seznam symbolů, které jsou externí k modulu - formát MODL0200 (*IMPORT),
- seznam jmen procedur a jejich typů - formát MODL0300 (*PROCLIST),
- seznam objeků, na něž se modul odkazuje při svém spojování do ILE programu nebo servisního programu - formát MODL0400 (*REFSYSOBJ),
- seznam informací o copyrightu - formát MODL0500 (*COPYRIGHT).

V programu DSPMODL byl zvolen výpis jmen všech modulů určité knihovny a jejich procedur (formát MODL0300) do uživatelského prostoru s následujícím tiskem do souboru QSYSPRT. Program DSPMODL má tři parametry – knihovna, výběr modulů, velikost uživatelského prostoru. Předávání parametrů je usnadněno stejnommenným CL příkazem DSPMODL.

Zdrojový text CL příkazu DSPMODL

```
CMD         PROMPT('Display list of modules')
            PARM        KWD(INPLIB) TYPE(*CHAR) LEN(10) MIN(1) +
                          CHOICE('*CURLIB, name') PROMPT('Input +
                          library')
            PARM        KWD(MODULE) TYPE(*CHAR) LEN(10) DFT(*ALL) +
                          CHOICE('*ALL, generic*, name') +
                          PROMPT('Module')
            PARM        KWD(SPCSIZ) TYPE(*DEC) LEN(9 0) DFT(20000) +
                          CHOICE('1 to 999999999') +
                          PROMPT('User space size')
```

Výsledné položky v uživatelském prostoru mají *proměnnou délku*, která je zapsaná na začátku každé položky. Další položku tedy získáme přičtením této délky, a ne délky zjištěné ze záhlaví prostoru.

Zdrojový text programu DSPMODL

```
     //*************************************************************
     //   Program DSPMODL - Tiskne modul(y) a seznam procedur
     //   Volá se CL příkazem DSPMODL (se třemi parametry)
     //*************************************************************
     Dcl-F QSYSPRT  Printer(132) OflInd(*InOA);

     //   Parametry pro funkci QBNLMODI (List Module Information)
     Dcl-DS modName qualified;
        module                  Char(10);
        inpLib                  Char(10);
     End-DS;

     //   Parametry pro vytvoření uživ. prostoru
     Dcl-DS spaceName;
        spcName                 Char(10) inz('MODLIST');
        spcLib                  Char(10) inz('QTEMP');
     End-DS;
     Dcl-S spaceSize             Int(10:0);
     Dcl-S spaceInit            Char(1) inz(X'00');
     Dcl-S extAttr              Char(10) inz('QBNLMODI');
     Dcl-S spaceAut             Char(10) inz('*ALL');
     Dcl-S spaceTxt             Char(50) inz(*BLANKS);
     Dcl-S spaceRepl            Char(10) inz('*YES');
     Dcl-S spaceDomain          Char(10) inz('*USER');

     //   Pracovní proměnné
```

```
       Dcl-S spacePtr           Pointer;
       Dcl-S listPtr            Pointer;
       Dcl-S procNamePtr        Pointer;
       Dcl-S procNameBased         Char(200) based(procNamePtr);
       Dcl-S procName           VarChar(50);
       Dcl-S modInfFmt             Char(8) Inz('MODL0300'); // Object information
format
       Dcl-S index                  Int(10:0);

       //   User Space Generic Header
       //   -------------------------
       Dcl-DS QUSH0100                 Based(SpacePtr); // Qus Generic Header 0100
          QUSUA                   Char(64) Pos(1);    // User Area
          QUSSGH                 BinDec(8:0) Pos(65);  // Size Generic Header
          QUSSRL                   Char(4) Pos(69);    // Structure Release Level
          QUSFN                    Char(8) Pos(73);    // Format Name
          QUSAU                    Char(10) Pos(81);   // Api Used
          QUSDTC                   Char(13) Pos(91);   // Date Time Created
          QUSIS                    Char(1) Pos(104);   // Information Status
          QUSSUS                 BinDec(8:0) Pos(105); // Size User Space
          QUSOIP                 BinDec(8:0) Pos(109); // Offset Input Parameter
          QUSSIP                 BinDec(8:0) Pos(113); // Size Input Parameter
          QUSOHS                 BinDec(8:0) Pos(117); // Offset Header Section
          QUSSHS                 BinDec(8:0) Pos(121); // Size Header Section
          QUSOLD                 BinDec(8:0) Pos(125); // Offset List Data
          QUSSLD                 BinDec(8:0) Pos(129); // Size List Data
          QUSNBRLE               BinDec(8:0) Pos(133); // Number List Entries
          QUSSEE                 BinDec(8:0) Pos(137); // Size Each Entry
          QUSSIDLE               BinDec(8:0) Pos(141); // CCSID List Ent
          QUSCID                   Char(2) Pos(145);   // Country ID
          QUSLID                   Char(3) Pos(147);   // Language ID
          QUSSLI                   Char(1) Pos(150);   // Subset List Indicator
          QUSERVED00               Char(42) Pos(151);  // Reserved
       End-DS;

       //   Format MODL0300
       //   ---------------
       Dcl-DS QBNL030001               Based(ListPtr); // Qbn LMODI MODL0300
          QBNES01                BinDec(8:0) Pos(1);   // Entry Size
          QBNMN04                  Char(10) Pos(5);    // modulePar Name
          QBNMLIBN01               Char(10) Pos(15);   // modulePar Library Name
          QBNPT                    Char(1) Pos(25);    // Procedure Type
          QBNERVED04               Char(3) Pos(26);    // Reserved
          QBNPNO00               BinDec(8:0) Pos(29);  // Procedure Name Offset
          QBNPNL00               BinDec(8:0) Pos(33);  // Procedure Name Length
          QBNPAO                   Char(10) Pos(37);   // Procedure ArgOpt
       End-DS;

       //   Data structure for Error Code Parameter QUSEC
       //   ---------------------------------------------
      /copy QSYSINC/QRPGLESRC,QUSEC

       //   Rozhraní programu DSPMODL (definice vstupních parametrů místo *ENTRY
PLIST)
       //   -------------------------
       Dcl-PI *N extpgm('DSPMODL');
          inpLibPar             Char(10);    // Input library
          modulePar             Char(10);    // *ALL, generic
          spaceSizePar        Packed(9:0);   // Space size
       End-PI;

       // Prototyp API QUSCRTUS
       Dcl-PR QUSCRTUS extpgm ('QUSCRTUS');
          spaceName              LikeDs(spaceName);
          extAttr                  Char(10)  ;
          spaceSize                 Int(10:0);
```

```
   spaceInit               Char(1)   ;
   spaceAut                Char(10)  ;
   spaceTxt                Char(50)  ;
   spaceRepl               Char(10)  ;
   QUSEC                  LikeDS(QUSEC);
   spaceDomain             Char(10)  ;
End-Pr QUSCRTUS;

// Prototyp API QUSPTRUS
Dcl-PR QUSPTRUS extpgm ('QUSPTRUS');
   spaceName               LikeDs(spaceName);
   spacePtr               Pointer;
End-PR QUSPTRUS;

// Prototyp API QBNLMODI
Dcl-PR QBNLMODI extpgm ('QBNLMODI');
   spaceName               LikeDs(spaceName);
   modInfFmt               Char(8)    ;
   modName                 Char(10)   ;
   QUSEC                  LikeDS(QUSEC);
End-PR QBNLMODI;

//=================================================================
//   Hlavní program
//=================================================================

//   Inicializace - získat User Space a seznam objektů
Exsr init;
//   Získat seznam modulů (QBNLMODI API)
Exsr getList;
//   Opakovat dokud není Information Status = C (complete)
//   (alespoň jednou)
DoU QUSIS = 'C';
   //   .Je-li Status = C nebo P (Partial) a počet položek je > 0 -
   //          - Zpracovat položky
   If QUSIS = 'C' or QUSIS = 'P'
   and QUSNBRLE > 0;
      //   ..Nastavit ukazatel na první položku
      listPtr = spacePtr + QUSOLD;
      //   ..Zpracovat daný počet položek
      For index = 1 To QUSNBRLE;
         //   ...Zpracovat položku
         Exsr procEntry;
         //   ...Zvýšit ukazatel o délku položky pro formát MODL0300
         listPtr += QBNES01;
      EndFor;
      //   ..Je-li Status = P (partial list) - Získat další část seznamu
      If QUSIS = 'P';
         Exsr getList;
      EndIf;

   EndIf;
EndDo;
//   Konec programu
Exsr end;

//-----------------------------------------------------------------
//   init - Jednorázová inicializace
//                     - Získat User Space a seznam modulů
//-----------------------------------------------------------------
BegSr init;

   // Přesunu parametry do pracovních proměnných
   modName.module = modulePar;
   modName.inpLib = inpLibPar;
   spaceSize = spaceSizePar;
```

```
     //   Vytvořit User Space v QTEMP
     callp QUSCRTUS ( spaceName:        //  Jméno User Space
                      extAttr:          //  Externí atribut
                      spaceSize:        //  Velikost User Space
                      spaceInit:        //  poč.hodnota
                      spaceAut:         //  autorizace
                      spaceTxt:         //  space text
                      spaceRepl:        //  replace
                      QUSEC:            //  error_code
                      spaceDomain  );
     //   Získat ukazatel spacePtr do User Space
     callp QUSPTRUS ( spaceName:
                      spacePtr   );      // ukazatel do User Space
  EndSr;

  //-----------------------------------------------------------------
  //   getList - Get program information list
  //-----------------------------------------------------------------
  BegSr getList;
     callp QBNLMODI ( spaceName:        //  Jméno User Space
                      modInfFmt:        //  Jméno formátu MODL0300
                      modName:          //  Jméno modulu
                      QUSEC     );      //  error_code
  EndSr;

  //-----------------------------------------------------------------
  //   procEntry - Zpracovat položku
  //-----------------------------------------------------------------
  BegSr procEntry;
     //   Získat ukazatel na jméno procedury
     procNamePtr = spacePtr + QBNPNO00;
     //   Jméno procedury pro tisk
     procName = procNameBased;
     %len(procName) = QBNPNL00;
     //   Tisknout položku
     Except objEntry;
  EndSr;

  //-----------------------------------------------------------------
  //   end - End routine
  //-----------------------------------------------------------------
  BegSr end;
     *inlr = *on;
     Return;
  EndSr;

  //=================================================================
  //   Printer output
  //=================================================================
oQSYSPRT   e            objEntry       1
  //   Jméno modulu
o                       QBNMN04             +2
  //   Jméno knihovny
o                       QBNMLIBN01          +2
  //   Typ procedury
o                       QBNPT               +2
  //   Jméno procedury
o                       procName            +2
  //   Délka jména procedury
o                       QBNPNL00      3     +2
```

Funkce typu UNIX a souvislost s jazykem C

Funkce tohoto typu jsou v dokumentaci popsány s ohledem na programování v jazyku C, protože v systémech typu UNIX je to hlavní programovací jazyk. Přesto však je možné je použít i v jiných jazycích, např. v ILE RPG nebo ILE COBOL, popř. ILE CL.

Chceme-li tyto funkce použít v jiném jazyku než ILE C, musíme si převést potřebné datové struktury a konstanty, jakož i prototypy a tvary volání z jazyka C do příslušného jazyka. K tomu je ovšem nutné se vyznat v jazyku C nebo si z různých zdrojů (na Internetu) opatřit potřebné popisy. Například na stránce https://www.scottklement.com/rpg/callc.html je fundovaný výklad o tom, jak postupovat při převodu do jazyka ILE RPG.

Ukázky převodu definic do jazyka ILE RPG

Příklad definice konstant pro funkci open() v jazyku C

```
/*   QSYSINC/H.FCNTL                                          */
#define O_RDONLY   00001        /* Open for reading only           */
#define O_WRONLY   00002        /* Open for writing only           */
#define O_RDWR     00004        /* Open for reading and writing    */
#define O_CREAT      00010      /* Create file if it doesn't exist */
#define O_EXCL       00020      /* Exclusive use flag              */
/*                   00040         reserved                       */
#define O_TRUNC      00100      /* Truncate flag                   */
```

Převod konstant do jazyka RPG

V pevném formátu:

```
      * File Access Modes
     D O_RDONLY        S             10I 0 Inz(X'01')
     D O_WRONLY        S             10I 0 Inz(X'02')
     D O_RDWR          S             10I 0 Inz(X'04')
     D O_CREATE        S             10I 0 Inz(X'08')
     D O_EXCL          S             10I 0 Inz(X'10')
     D O_TRUNC         S             10I 0 INZ(X'40')
```

Ve volném formátu:

```
Dcl-S O_RDONLY        Int(10:0) Inz(X'01');
Dcl-S O_WRONLY        Int(10:0) Inz(X'02');
Dcl-S O_RDWR          Int(10:0) Inz(X'04');
Dcl-S O_CREATE        Int(10:0) Inz(X'08');
Dcl-S O_EXCL          Int(10:0) Inz(X'10');
Dcl-S O_TRUNC         Int(10:0) INZ(X'40');
```

Příklad prototypu funkce open() v jazyku C

```
int open(const char *path, int oflag, . . .);
```

Převod prototypu do jazyka RPG

V pevném formátu:

```
      *   Open stream file
     D Open            Pr            10I 0 ExtProc('open')
     D  PathP                          *   Value Options(*String)
     D  Oflag                        10I 0 Value
     D  Mode                         10U 0 Value Options(*Nopass)
     D  Codepage                     10U 0 Value Options(*Nopass)
```

Ve volném formátu:

```
// Open stream file
Dcl-PR Open Int(10:0) ExtProc('open');
   PathP         Pointer     Value Options(*String);
```

```
   Oflag           Int(10:0) Value;
   Mode            Uns(10:0) Value Options(*Nopass);
   Codepage        Uns(10:0) Value Options(*Nopass);
End-PR;
```

Parametry Mode a Codepage bylo nutno doplnit až po prostudování příslušné kapitoly v dokumentaci, z níž plyne, že za tři tečky lze dosadit až dva nepovinné - *Options(*Nopass)* -parametry, které mají předepsaný význam. První parametr (PathP) je tzv. *null terminated string*, tedy znakový řetězec ukončený nulovým bajtem X'00' (v jazyku C označovaným \0). Proto je uvedeno *Options(*String)*. Význam parametru Oflag je ten, že obsahuje bitové příznaky, které se získají přičítáním (v jazyku C spíše bitovým součtem OR).

Například konkrétní volání funkce open() k vytvoření a otevření binárního souboru by v jazyku C mohlo vypadat takto:

```
fdout = open ( outpath, O_CREAT | O_RDWR | O_TRUNC + O_CCSID + O_TEXTDATA,
                        S_IRWXU | S_IRWXG | S_IRWXO);
```

Zde je poslední parametr (codepage) vynechán. Je vidět, že převod z jazyka C do jiného jazyka není přímočarý, ale vyžaduje značné úsilí a pečlivost.

Převod volání do jazyka RPG

V pevném formátu:

```
C                   Eval      Mode  = S_IRWXU + S_IRWXG + S_IRWXO
C                   Eval      Oflag = O_CREATE + O_RDWR + O_TRUNC +
C                                     O_CCSID + O_TEXTDATA

C                   Eval      FDOut = Open(%TrimR(OutPath): Oflag: Mode)
```

Ve volném formátu:

```
Mode  = S_IRWXU + S_IRWXG + S_IRWXO;  // pro uživatele, skupinu a ostatní
Oflag = O_CREATE + O_RDWR + O_TRUNC + // vytvořit, čtení i zápis, smazat data
        O_CCSID + O_TEXTDATA;         // kódová sránka, textová data

FDOut = open(%trimr(OutPath): oflag: mode: CCSID);
```

Získání popisů a prototypů pro jazyk RPG

V programech psaných ve volném formátu lze přikopírovat zdrojové členy zapsané v *pevném* formátu pomocí direktivy /COPY nebo /INCLUDE.

V dodatku je uveden opis zdrojového členu PROTOTYPY, který obsahuje vybrané prototypy některých funkcí a k nim patřící data *pevném* formátu. Tento člen používáme v našich příkladech.

Alternativně lze použít direktivu pro předkompilátor

/COPY QSYSINC/QRPGLESRC,IFS

který obsahuje všechny popisy a prototypy pro IFS, také v *pevném* formátu.

Práce se soubory v IFS

Soubory v IFS se nazývají proudové (stream file), protože jejich obsahem je proud bajtů, o jejichž struktuře není operačnímu systému nic známo (na rozdíl od databázových souborů). V integrovaném systému souborů (IFS - Integrated File System) je každému souboru přiděleno pořadové číslo zvané *file descriptor*, které se získává při otevírání funkcí *open* a který nahrazuje jméno souboru. Další příkazy *write, read, close* se na file descriptor odvolávají. Příkazů je pochopitelně mnohem více, ale zde použijeme jen ty vyjmenované.

Příkaz *Open* jsme už probrali.

Příkaz *Close* uzavírá IFS soubor.

```
     *   int close(int descriptor)

    D Close           Pr                  ExtProc('close')
    D                                10I 0 Value
```

Příkaz *Read* čte data z IFS souboru.

```
     *   ssize_t read(int descriptor,
     *                void *buffer,
     *                size_t buffer_length);

    D Read            Pr            10I 0 Extproc('read')
    D                               10I 0 Value
    D                                 *   Value
    D                               10U 0 Value
```

Příkaz *Write* zapisuje data do IFS souboru.

```
     *   ssize_t write(int file_descriptor,
     *                 const void *buffer,
     *                 size_t buffer_length);

    D Write           Pr            10I 0 ExtProc('write')
    D                               10I 0 Value
    D                                 *   Value
    D                               10U 0 Value
```

Funkce typu UNIX jsou zpřístupněny systémovou spojovací knihovnou (binding directory) QC2LE, kterou zadáme v příkazu CTL-OPT zároveň s aktivační skupinou, která nesmí být předvolená (default). Tyto podmínky jsou znakem konceptu ILE (Integrated Language Environment).

```
Ctl-Opt dftActGrp(*no) actGrp(*new) bnddir('QC2LE');
```

Příkaz `actGrp(*new)` je určen pro ladění, aby se nemusela zvlášť ukončovat aktivační skupina neb úloha.

Poznámka: Ve spojovacím adresáři (binding directory) QC2LE jsou obsaženy servisní programy potřebné k práci s funkcemi v jazyku C a C++. Ty jsou již od verze 6.1 jazyka RPG součástí systémového spojovacího adresáře. Příkaz `bnddir('QC2LE')` tedy není nutné zapisovat.

Vytvoření IFS souboru se zadanou kódovou stránkou

Program CRTIFSF vytváří v zadaném adresáři textový soubor se zadaným znakovým kódem. Volá se CL příkazem stejného jména, v němž se zadají dva parametry, cesta k souboru a číslo CCSID.

Program je zapsán jako tzv. lineární, tj. bez cyklu RPG, což je dáno klíčovým slovem MAIN v příkazu CTL-OPT. V tom případě musí být celý program ohraničen příkazy DCL-PROC a

END-PROC jako podprocedura. V rozhraní programu DCL-PI se pak místo jména programu zapíše *N a klíčové slovo EXTPGM.

Funkce open() připraví vytvářený soubor a funkce close() vytvoření dokončí. Dojde-li ve funkci open() k chybě, vrátí místo deskriptoru souboru číslo -1. V tom případě vypíše program obsah paměti s informacemi o chybě a skončí.

Program CRTIFSF

```
**free
//*********************************************************************
//   Program CRTIFSF
//   ---------------
//   Program, který vytvoří prázdný IFS soubor v zadaném adresáři
//   se zadanou kódovou stránkou (CCSID)
//   Volá se CL příkazem CRTIFSF
//*********************************************************************
Ctl-Opt dftActGrp(*no) actGrp(*new) bnddir('QC2LE')
        MAIN(CRTIFSF);

Dcl-PROC CRTIFSF;

   //   Prototypy a data pro funkce IFS
   /copy QRPGLESRC,PROTOTYPY

   //  Rozhraní programu
   Dcl-PI *N ExtPgm;
      path                     Char(256);
      CCSID                     Int(10:0);
   End-PI;

   //  příznaky pro funkci open
   Dcl-S oflag                  Int(10:0) inz(0);
   Dcl-S mode                   Int(10:0) inz(0);

   //  deskriptor souboru (vznikne ve funkci open)
   Dcl-S fdOut                  Int(10:0) inz(0);

   // určím přístupová práva čtení, zápisu, provedení
   mode  = S_IRWXU + S_IRWXG + S_IRWXO;   // pro uživatele, skupinu a ostatní

   // určím příznaky pro výstupní soubor
   oflag = O_CREATE + O_RDWR + O_TRUNC +  // vytvořit, čtení i zápis, smazat data
           O_CCSID + O_TEXTDATA;          // kódová sránka, textová data

   path = %trimr(path) + x'00';

   // pokusím se otevřít výstupní soubor
   fdOut = open(%trimr(Path): oflag: mode: CCSID);

   // nepodaří-li se soubor otevřít, zjistím chybu
   If fdOut = -1;
      ErrNoP = GetErrNo;
      ErrMsgP = StrError(ErrNo);
      Dump(a) 'chyba CRTIFSF';
      *inlr = *on;
      Return;
   EndIf;

   // v případě úspěchu uzavřu výstupní soubor
   CallP close(fdOut);

End-PROC CRTIFSF;
```

CL příkaz CMDIFSF

```
CMD
PARM       KWD(FILEPATH) TYPE(*CHAR) LEN(256) +
             DFT('/home/newFile.txt') +
             PROMPT('Path to the IFS file to create')
PARM       KWD(CCSID) TYPE(*UINT4) DFT(870) +
             PROMPT('CCSID of the IFS file')
```

Kopírování "save" souboru do IFS souboru

Program COPY2 kopíruje save file SAVEFILE do proudového souboru (stream file) do IFS adresáře *home*. *Vstupní* soubor SAVEFILE (v libovolné knihovně zapsané v *LIBL) je objekt typu *FILE se záznamy dlouhými 528 bajtů vytvořený předem třeba příkazem SAVLIB. Je zvláštní tím, že nemá žádný člen, ale obsahuje data. *Výstupní* IFS soubor je */home/savefile*.

V programu nejsou ošetřeny chyby vstupu a výstupu.

Zdrojový text programu COPY2

```
**free
Ctl-Opt DFTACTGRP(*NO) ACTGRP('QILE') BNDDIR('QC2LE');

Dcl-F SAVEFILE DISK(528); // Vstupní soubor

Dcl-DS Data Len(528);     // Datová struktura bez podpolí
End-DS;

/COPY QRPGLESRC,PROTOTYPY

Dcl-S OutPath               Char(256) Inz('/home/savefile');

Dcl-S DataL               Packed(5:0) Inz(%Len(Data));
Dcl-S DataLenBin             Int(10:0) Inz(%Len(Data));
Dcl-S DataAdr             Pointer Inz(%Addr(Data));

Dcl-S Oflag                  Int(10:0) Inz(0);
Dcl-S Mode                   Int(10:0) Inz(0);
Dcl-S FDOut                  Int(10:0) Inz(0);
Dcl-S Size                   Int(10:0) Inz(0);

Mode  = S_IRWXU;
Oflag = O_CREATE + O_RDWR;
FDOut = Open(%TrimR(OutPath): Oflag: Mode);

Read SAVEFILE Data;
DoW Not %EOF;
   Size = Write(FDOut: DataAdr: DataLenBin);
   Read SAVEFILE Data;
EndDo;

CallP Close(FDOut);

*InLR = *On;
Return;
```

Kopírování z IFS souboru zpět do "save" souboru

Program COPY2A kopíruje proudový soubor (stream file) z IFS adresáře *home* do save souboru SAVEFILE2 v knihovně VZAPI_FREE. Vstupní IFS soubor je /home/savefile, jehož obsah je binární obraz nějakého save souboru, třeba SAVEFILE. Výstupní soubor SAVEFILE2 v knihovně VZAPI_FREE je objekt typu *FILE se záznamy dlouhými 528 bajtů a musí být *vytvořen předem* (třeba prázdný) příkazem CRTSAVF. Tento soubor nemá žádný

člen, ale přesto může obsahovat data. Pro zápis dat ale nelze použít příkaz WRITE. Lze použít příkaz EXCEPT s popisem výstupu.

Zdrojový text programu COPY2A

```
     Ctl-Opt DFTACTGRP(*NO) ACTGRP('QILE') BNDDIR('QC2LE');

     Dcl-F SAVEFILE2 DISK(528) Usage(*Output);

    /COPY QRPGLESRC,PROTOTYPY

     Dcl-S InPath               Char(256) Inz('/home/savefile');

     Dcl-S Data                 Char(528);
     Dcl-S DataL              Packed(5:0) Inz(%Len(Data));
     Dcl-S DataLenBin            Int(10:0) Inz(%Len(Data));
     Dcl-S DataAdr            Pointer Inz(%Addr(Data));

     Dcl-S Oflag                 Int(10:0) Inz(0);
     Dcl-S Mode                  Int(10:0) Inz(0);
     Dcl-S FDIn                  Int(10:0) Inz(0);
     Dcl-S Size                  Int(10:0) Inz(0);

        Mode  = S_IRWXU;
        Oflag = O_RDONLY;
        FDIn  = Open(%TrimR(InPath): Oflag: Mode);

        Size = Read(FDIn: DataAdr: DataLenBin);
        DoW Size = DataLenBin;
           Except Output;
           Size = Read(FDIn: DataAdr: DataLenBin);
        EndDo;

        CallP Close(FDIn);
        *InLR = *On;
        Return;

      // Do výstupního souboru nelze psát příkazem WRITE
    OSAVEFILE2 E            Output
    O                       Data
```

Komunikace v TCP/IP pomocí soketů

Pod slovem soket (socket = zásuvka) si můžeme představit určité místo v počítači, které slouží ke komunikaci s jinými počítači a přes něž proudí data. Podobně jako proudovému souboru v IFS odpovídá pořadové číslo zvané file descriptor, odpovídá soketu pořadové číslo zvané *socket descriptor*. Ke komunikaci v TCP/IP slouží řada funkcí (příkazů) typu UNIX, jejichž prototypy jsou převedeny z jazyka C do jazyka ILE RPG.

Prototypy a data potřebných soketových funkcí

Funkce Socket

Funkce *Socket* vytváří a otevírá soket a má tento prototyp:

```
int socket(int address_family,
           int type,
           int protocol)
```

```
D Socket          Pr            10I 0 Extproc('socket')

D                               10I 0 Value
D                               10I 0 Value
D                               10I 0 Value
```

Parametry volání mohou být voleny z následujících definic konstant:

```
 * Address families
D AF_UNIX         C                   1
D AF_INET         C                   2
D AF_NS           C                   6
D AF_TELEPHONY    C                   99

 * Socket types
D SOCK_STREAM     C                   1
D SOCK_DGRAM      C                   2
D SOCK_RAW        C                   3
D SOCK_SEQPACKET  C                   5
D SOL_SOCKET      C                   -1
```

Zde použijeme tyto parametry:

- *address_family:* AF_INET,
- *type:* SOCK_STREAM,
- *protocol:* 0 (dosadí se automaticky ten protokol, který odpovídá typu soketu).

Funkce GetHostByName

Příkaz *GetHostByName* získává IP adresu v binární formě pro dané jméno počítače (host name) a má tento prototyp:

```
struct HostEnt *gethostbyname(char *host_name);
```

```
D GetHostByName   Pr              *   Extproc('gethostbyname')
D                                 *   Value
```

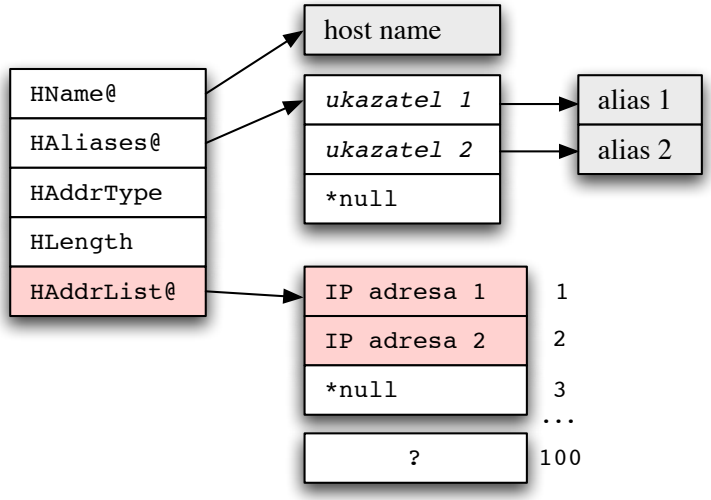
IP adresa je potřebná pro příkaz *Connect*. Výsledkem příkazu *GetHostByName* je *ukazatel* na strukturu *HostEnt* (host entries), z níž můžeme získat binární IP adresu.

```
struct HostEnt {
   char   *h_name;
   char   **h_aliases;
   int    h_addrtype;
   int    h_length;
   char   **h_addr_list;
};
```

V jazyku C je struktura zapsána kompaktně, protože obsahuje i ukazatele na ukazatele (dvě hvězdičky před jménem). V jazyku ILE RPG musíme ukazatele druhé úrovně definovat zvlášť:

```
D HostEnt         DS                  Align Based(HostP)
D  HName          P                     *
D  HAliasesP                            *
D  HAddrType                         10I 0
D  HLength                           10I 0
D  HAddrListP                           *

D  HAddrList      S                     *   Based(HAddrListP) Dim(100)
```

Alernativní jména (alias) nebudeme zjišťovat. Položka *HAddrList*P představuje ukazatel na seznam ukazatelů *HAddrList*. Tento seznam obsahuje ukazatele na binární IP adresy. Je ukončen nulovým ukazatelem **null*.



Jména počítačů (host name a alias) jsou znakové řetězce ukončené znakem X'00'.

Funkce Connect

Funkcí *Connect* zahajuje klient spojení se serverem, který jej může přijmout funkcí *Accept*.

```
int connect(int socket_descriptor,
            struct sockaddr *destination_address,
            int address_length);
```

```
D Connect         Pr            10I 0 Extproc('connect')
D                               10I 0 Value
D                                 *   Value
D                               10I 0 Value
```

První parametr je pořadové číslo soketu (deskriptor) získané příkazem *Socket*.

Druhý parametr je ukazatel na datovou strukturu reprezentující IP adresu a port:

```
D SocketAddr      DS
D   SinFamily                    5I 0
D   SinPort                      5U 0
D   SinAddr                     10U 0
D   SinZero                      8A   Inz( *ALLX'00' )

D SockAddr        S                   Inz( %Addr(SocketAddr) )
```

Třetí parametr je délka této struktury:

```
D AddressLength   S             10I 0
```

```
D AddrLen         S               *   Inz( %Addr(AddressLength) )
```

Funkce Bind

Funkci *Bind* použije server, aby připojil program k soketu.

```
int bind(int socket_descriptor,
         struct sockaddr *local_address,
         int address_length)
```

```
D Bind            Pr            10I 0 ExtProc('bind')
D                               10I 0 Value
D                                 *   Value
D                               10I 0 Value
```

Parametry jsou stejné jako u příkazu *Connect*.

Funkce Listen

Funkcí *Listen* server dovoluje klientskému programu, aby se připojil. Zároveň stanoví počet klientů, které se mohou připojit (parametr back_log).

```
int listen(int socket_descriptor,
           int back_log);
```

```
D Listen          Pr            10I 0 ExtProc('listen')
D                               10I 0 Value
D                               10I 0 Value
```

Funkce Accept

Funkcí *Accept* server přijímá klientský příkaz *Connect* a vytvoří nový soket pro komunikaci s klientem.

```
int accept(int socket_descriptor,
           struct sockaddr *address,
           int *address_length);
```

```
D Accept          Pr            10I 0 ExtProc('accept')
D                               10I 0 Value
D                                 *   Value
D                                 *   Value
```

Funkce Read, Write a Close

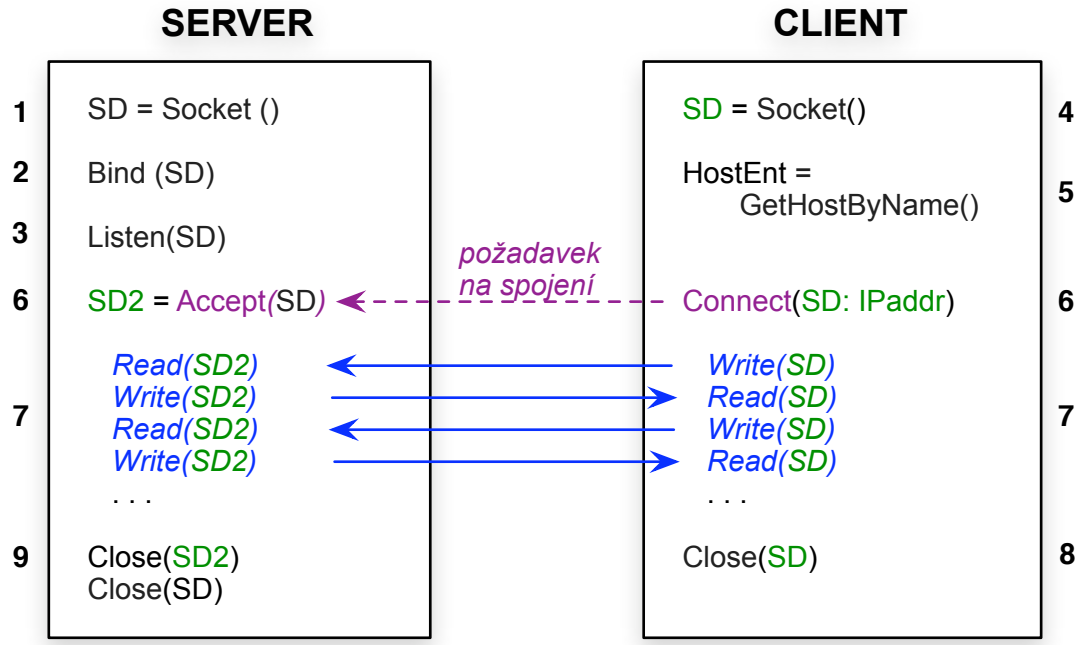
Funkce *Read*, *Write* a *Close* jsou shodné se stejnojmennými příkazy pro proudové soubory v IFS. Jen místo čísla *file description* je použito číslo *socket description*, což je prakticky totéž.

Příklad - sokety

Příklad se skládá ze dvou programů - SERVER a CLIENT. Klient zahajuje komunikaci požadavkem, server na požadavek odpovídá. Server najde údaje o zboží v souboru ZBOZI na základě požadavku, který přijme ze zprávy od klienta. Příklad je zjednodušený natolik, aby pouze ilustroval použití soketových funkcí.

Server může být spuštěn v dávkové (batch) nebo interakční úloze. Klient může být spuštěn jen jednou a když skončí, uzavře komunikační soket (funkcí *Close*) a způsobí tak havarijní ukončení serveru (ve funkci *Read*). Klient může řádně ukončit server zasláním textu END jako “čísla zboží”.

Zjednodušené schema komunikace mezi servererm a klientem a časovou posloupnost příkazů (bez databázového a obrazovkového souboru) ukazuje následující obrázek.



Program SERVER

Program SERVER přijme požadavek na spojení (*Connect*) od programu CLIENT příkazem *Accept*, přečte číslo zboží od klienta příkazem *Read* a najde je v databázi. Údaje o zboží pošle zpět klientovi příkazem *Write* a čeká na další požadavek od klienta v příkazu *Read*. Program není schopen komunikovat s více než jedním klientem. I kdyby funkce *Listen* připouštěla více klientů, nestačilo by to. Paralelní zpracování více klientů v serveru je složitější.

```
**free
Ctl-Opt Debug DFTACTGRP(*NO) ACTGRP('QILE') BNDDIR('QC2LE');

//   Soubor ZBOZI
Dcl-F ZBOZI                   Keyed;

Dcl-DS SocketData                ExtName('ZBOZI') qualified;
End-DS;
Dcl-S SocketDataP        Pointer Inz(%Addr(SocketData));
Dcl-S SockDtaLen             Int(10:0) Inz(%size(SocketData));
Dcl-S PortNumber             Int(10:0) Inz(3005);
//   Cislo "socket description" pro server
Dcl-S SD                     Int(10:0);
//   Cislo "socket description" pro klienta
Dcl-S SD2                    Int(10:0);
//   Navratovy kod pro sokety
Dcl-S RC                     Int(10:0);

//   Prototypy podprocedur pro sokety a potrebne popisy dat
/COPY QRPGLESRC,PROTOTYPY

//   Prototyp procedury pro obsluhu chyb
Dcl-PR ErrorHdlr;
   DumpText                 Char(12) Value;
End-PR;


//   Ziskat "socket descriptor" (server socket) pro sebe
SD = Socket (AF_INET: SOCK_STREAM: 0 );

//   Je-li chyba ve funkci socket - dump, zavřít soket, ukoncit program
If SD < 0;
```

```
   CallP ErrorHdlr ('ServerSocket');
   Return;
EndIf;

//   Pripojit soket k IP adrese (bind)
SocketAddr = *ALLX'00';
SinFamily  = AF_INET;
SinPort    = PortNumber;
SinAddr    = INADDR_ANY;
RC = Bind (SD: %ADDR(SocketAddr) : %SIZE(SocketAddr));
//   Je-li chyba - dump, zavřít soket, ukoncit program.
//   Chyba nastává, když už byl předtím soket připojen.
If RC < 0;
   CallP ErrorHdlr ('ServerBind');
   Return;
EndIf;

//   Naslouchat (listen) 1 klientovi
RC = Listen (SD: 1);

//   Je-li chyba - dump, zavřít soket, ukoncit program
If RC < 0;
   CallP ErrorHdlr ('ServerListen');
   Return;
EndIf;

//   Prijmout prichozi pozadavek spojeni (connect) od klienta.
//   Pro klienta se vytvori novy soket (SD2).
SD2 = Accept (SD: SockAddr: AddrLen);

//   Je-li chyba - dump, zavřít soket, ukoncit program
If RC < 0;
   CallP ErrorHdlr ('ServerAccept');
   Return;
EndIf;

//   Cyklus write/read
DoW 0 = 0;

   //   Precist data z klientova soketu do promenne SocketData
   RC = Read (SD2: SocketDataP: %len(SocketData.CZBOZI));

   //   Je-li chyba - dump, zavřít soket, ukoncit program
   If RC <= 0;
      CallP ErrorHdlr ('ServerRead');
      Return;
   EndIf;

   //   Jsou-li prvni znaky cisla zbozi END - ukoncit server
   If SocketData.CZBOZI  = 'END';
      Leave;
   EndIf;

   //   Cist odpovidajici zaznam ze souboru ZBOZI podle klice
   Chain  SocketData.CZBOZI  ZBOZI;

   //   Nebyl-li zaznam nalezen - dosadit nahradni data
   If Not %Found;
      CENAJ = 0;
      POPIS = *All'?';
   EndIf;

   // Presun dat získaných z databaze do datové struktury
   SocketData.CENAJ = CENAJ;
   SocketData.POPIS = POPIS;
```

```
   //   Poslat odpoved klientovi
   RC = Write (SD2: SocketDataP: SockDtaLen);

   //   Je-li chyba - dump, zavřít soket, ukoncit program
   If RC <= 0;
      CallP ErrorHdlr ('ServerWrite');
      Return;
   EndIf;

   //   Konec cyklu write/read
EndDo;

//   Konec programu
CallP Close(SD2);
CallP Close(SD);
*InLR = *ON;
Return;

//----------------------------------------------------------------------
//   ErrorHdlr - Podprocedura "Error handling" -
//               tiskne vypis pameti (dump).
//               Vstupni parametr je text k rozeznani vypisu.
//----------------------------------------------------------------------

Dcl-Proc ErrorHdlr;

    Dcl-PI ErrorHdlr;
       DumpText                 Char(12) Value;
    End-PI;

    ErrNoP = GetErrNo;
    ErrMsgP = StrError(ErrNo);

    Dump DumpText;
    CallP Close(SD);
    *InLR=*On;

End-Proc ErrorHdlr;
```

Program CLIENT

Program CLIENT vyzve uživatele, aby zadal číslo zboží. Po stisku Enter se spojí s programem SERVER, pošle mu číslo zboží, přečte odpověď od serveru (údaje o zboží) a zobrazí ji na obrazovce. Stiskne-li uživatel klávesu Enter, vyzve znovu uživatele k zadání čísla zboží. Po stisku klávesy F3 program uzavře soket a skončí. K řádnému ukončení serveru může uživatel zadat text END místo čísla zboží.

```
**free
Ctl-Opt DEBUG DFTACTGRP(*NO) ACTGRP('QILE') ~~BNDDIR('QC2LE')~~;

//   Obrazovkovy soubor k vyzve a zobrazeni dat ze serveru
Dcl-F ZBOZIW         WORKSTN ;

Dcl-DS SocketData            ExtName('ZBOZI') qualified;
End-DS;

Dcl-S SockDtaLen             Int(10:0) Inz (%Size(SocketData));
Dcl-S PortNumber             Int(10:0) Inz(3005);
//   Cislo "socket description" pro klienta
Dcl-S SD                     Int(10:0);
//   Navratovy kod pro sokety
Dcl-S RC                     Int(10:0);

//   Jmeno serveru (pocitace, hosta) a ukazatele
```

```
Dcl-S ServerName            Char(255) Inz('LOCALHOST');
Dcl-S ServerP             Pointer Inz;
Dcl-S HostEntP            Pointer Inz;

//   Prototypy podprocedur pro sokety a potrebne popisy dat
/INCLUDE QRPGLESRC,PROTOTYPY

//   Prototyp procedury pro obsluhu chyb
Dcl-PR ErrorHdlr;
   DumpText                 Char(12) Value;
End-PR;


//   Ziskat "socket descriptor"
SD = Socket( AF_INET : SOCK_STREAM : 0);

//   Je-li chyba ve funkci socket - dump, zavřít soket, ukoncit program
If SD < 0;
   CallP ErrorHdlr ('ClientSocket');
   Return;
EndIf;

//   Vyplnit potrebna pole ve strukture IP addresy
SocketAddr = *ALLX'00';
SinFamily = AF_INET;
SinPort = PortNumber;

//   Pripravit jmeno serveru pro funkci GetHostByName
ServerName = %Trim(ServerName) + X'00';
ServerP = %Addr(ServerName);

//   Ziskat adresu serveru, je-li dano jmeno serveru
HostEntP = GetHostByName(ServerP);

//   Nelze-li ziskat adresu - dump, zavřít soket, ukoncit program
If HostEntP = *NULL;
   CallP ErrorHdlr ('ClientHostN');
   Return;
EndIf;

//   Kopirovat IP addresu ze struktury "host entry" do struktury IP addresy serveru
HAddrP = HAddrList(1);
SinAddr = HAddr;

//   Pripojit se k serveru
RC = Connect( SD:
     %Addr(SocketAddr) :
     %Size(SocketAddr)  );
//   Je-li chyba - dump, zavřít soket, ukoncit program
If RC < 0;
   CallP ErrorHdlr ('ClientConnect');
   Return;
EndIf;

//   Cyklus write/read
DoW 0 = 0;

   //   Vyzva uzivateli, aby vlozil cislo zbozi
   Exfmt ZBOZIW0;

   //   F3 - opustit cyklus a ukoncit program
   If *In03;
      Leave;
   EndIf;

   //   Poslat cislo zbozi do serveru pres socket
```

```
   RC = Write( SD : %Addr(CZBOZI) : %len(CZBOZI) );

   //   Byla-li chyba - dump, zavřít soket, ukoncit program
   If RC < 0;
      CallP ErrorHdlr ('ClientWrite');
   EndIf;

   //   Byl li vstup z obrazovky END - ukoncit program
   //   (zpusobi take ukonceni serveru)
   If CZBOZI = 'END';
      Leave;
   EndIf;

   //   Precist odpoved ze serveru
   RC = Read ( SD : %Addr(SocketData) : SockDtaLen );

   //   Je-li chyba - dump, uzavřít soket, ukoncit program
   If RC < 0;
      CallP ErrorHdlr ('ClientRead');
      Leave;
   EndIf;

   //   Nejsou-li zadna data ze serveru - ukoncit server, dump
   If RC = 0;
      CallP ErrorHdlr ('ClientRead2');
      Leave;
   EndIf;

   //   Zobrazit data prectena ze serveru
   CENAJ = SocketData.CENAJ;
   POPIS = SocketData.POPIS;
   Exfmt ZBOZIW1;

   //   F3 - ukoncit program
   If *In03;
      Leave;
   EndIf;

   //   Konec cyklu write/read
EndDo;

//   Konec programu
CallP Close(SD);
*InLR=*On;
Return;

//-----------------------------------------------------------------------
//   ErrorHdlr - Podprocedura "Error handling" -
//               tiskne vypis pameti (dump).
//               Vstupni parametr je text k rozeznani vypisu.
//-----------------------------------------------------------------------

Dcl-Proc ErrorHdlr;

    Dcl-PI ErrorHdlr;
       DumpText                Char(12) Value;
    End-PI;

    ErrNoP = GetErrNo;
    ErrMsgP = StrError(ErrNo);

    Dump DumpText;
    CallP Close(SD);
    *InLR=*On;

End-Proc ErrorHdlr;
```

Ošetření chyb v programech SERVER a CLIENT

Pro zjištění chyb v programech SERVER a CLIENT je použito analogie s programy psanými v jazyku C, tedy funkce errno() a strerror(). Funkce errno() je použita v podobě __errno(). Prototypy a data pro tyto funkce:

```
* Get error number

     *   extern int * __errno(void);

    D GetErrNo        Pr              *   ExtProc('__errno')


     * Get error text

     *   char *strerror(int errnum);

    D StrError        Pr              *   ExtProc('strerror')
    D                                10I 0 Value

     * Error related data

    D ErrNo           S               10I 0 Based(ErrNoP)
    D ErrNoP          S                 *   Inz
    D ErrMsg          S               60A   Based(ErrMsgP)
    D ErrMsgP         S                 *   Inz
```

Zdroje DDS pro program CLIENT a SERVER

Databázový soubor ZBOZI použitý v programu SERVER

```
A                                       UNIQUE
A         R ZBOZIR
 *   Cislo zbozi
A           CZBOZI         5            COLHDG('Cislo' 'zbozi')
 *   Jednotkova cena
A           CENAJ          9  2         COLHDG('Jednotkova' 'cena')
 *   Popis zbozi
A           POPIS         50            COLHDG('Popis zbozi')

 *   Klicove pole
A         K CZBOZI
```

Obrazovkový soubor ZBOZIW použitý v programu CLIENT

```
A                                       DSPSIZ(24 80 *DS3)
A                                       REF(ZBOZI)
A                                       CA03(03 'Konec')
 *   Format k vlozeni cisla zbozi
A         R ZBOZIW0
A                                    3  2'Zadejte cislo zbozi a stisknete En-
A                                       ter.'
A                                       DSPATR(HI)
A                                    5  2'Cislo zbozi....:'
A           CZBOZI    R        B  5 20
A                                   23  2'F3=Konec'
A                                       COLOR(BLU)
 *   Format k zobrazeni dat zaznamu
A         R ZBOZIW1
A                                    5  2'Cislo zbozi....:'
A                                       DSPATR(HI)
A           CZBOZI    R        O  5 20
A                                    6  2'Jednotkova cena:'
A                                       DSPATR(HI)
```

```
A            CENAJ     R        O  6 20EDTCDE(K)
A                                  7  2'Popis zbozi.....:'
A                                      DSPATR(HI)
A            POPIS     R        O  7 20
A                                 23  2'F3=Konec'
A                                      COLOR(BLU)
A  80                             24  2'Server neodpovida'
A                                      DSPATR(HI)
```

Dodatek

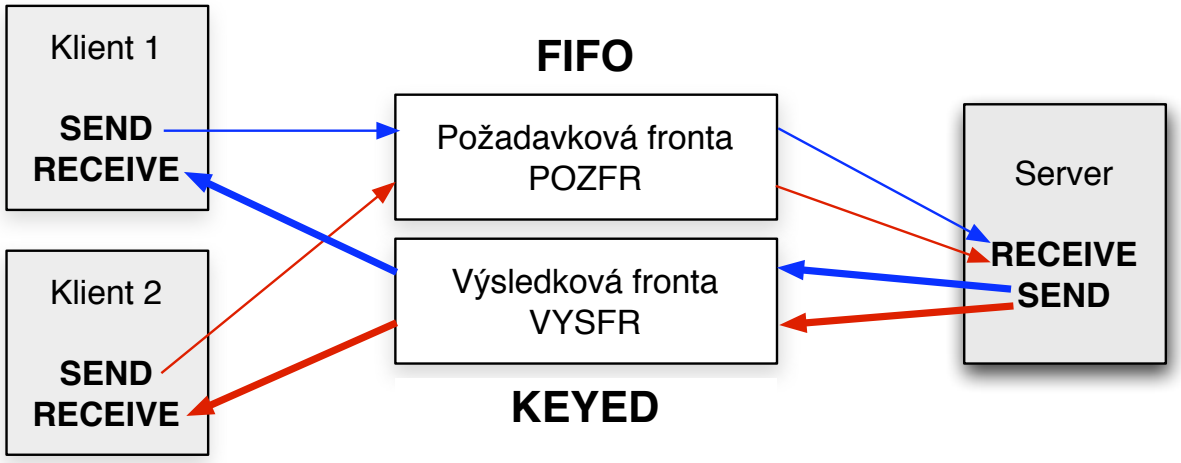
Datové fronty

Datová fronta (data queue) je objekt typu *DTAQ a vytváří se příkazem **CRTDTAQ**. Datové fronty se používají ke komunikaci mezi programy, zvláště je-li některý z nich provozován v dávkovém režimu (*BATCH). Ke komunikaci lze ovšem použít i jiné objekty, např. databázový soubor nebo frontu zpráv. Fronta zpráv však nefunguje dobře u dávkově spuštěných programů. Datové fronty nespotřebují tolik paměti jako databáze a komunikace s nimi probíhá rychleji. Existují tři typy datových front. Fronta typu *FIFO* (first in, first out), tedy obyčejný seznam, fronta typu *LIFO* (last in, first out), tedy zásobník, a fronta typu *KEYED* s přístupem podle klíče.

V příkladech si probereme frontu FIFO a KEYED. Vytvoříme dvě fronty, přes něž budou komunikovat programy SERVERR a KLIENTR.

Jedna fronta - POZFR - bude požadavková, přes niž bude posílat zprávy (záznamy) klient serveru a server si je z ní bude odebírat v tom pořadí, jak do fronty přišly - FIFO.

Druhá fronta - VYSFR - bude výsledková, přes niž server posílá server zprávy (záznamy) o výsledku své činnosti klientovi a klient si je odtud odebírá podle klíče. Klíčem fronty VYSFR je *identifikace úlohy* (číslo, jméno, uživatel), ve které běží klient, který si vyžádal službu u serveru. Identifikaci úlohy předává už klient jako součást své zprávy do požadavkové fronty, odkud ji server zkopíruje do odpovědi jako klíč výsledkové fronty. Klient si pak přečte jen tu výsledkovou zprávu, která je určena jemu.



Uspořádání s dvěma frontami - FIFO a KEYED - umožňuje provozovat více klientů a jeden server, popř. i více serverů. Servery si čtou z požadavkové fronty tak, že zprávu přečte ten, který se k ní dostane dřív. V našem příkladu se přečtením zpráva z fronty odstraní. Do výsledkové fronty pak umisťují zprávy v nahodilém pořadí.

Funkce API týkající se datových front jsou popsány v části *Object APIs - Data Queues*:

QCLRDTAQ) - smaže všechny záznamy z fronty,
QRCVDTAQ - čte záznam z fronty (přečtený záznam je z fronty odstraněn),
QSNDDTAQ - zapíše záznam do fronty,
QMHQRDQD - zjistí informace o frontě,
QMHRDQM - čte záznam z fronty a neodstraní jej.

V našem příkladu použijeme jen první tři funkce.

Vytvoření datových front

Datové fronty POZFR a VYSFR vytvoříme CL příkazem *CRTDTAQ*.

```
/*   Existuje fronta POZFR v *LIBL?                                    */
             CHKOBJ     OBJ(*LIBL/POZFR) OBJTYPE(*DTAQ)
/*   Jestliže neexistuje, vytvořit ji v *CURLIB                        */
             MONMSG     MSGID(CPF9801) EXEC(DO)
CRTDTAQ    DTAQ(*CURLIB/POZFR) TYPE(*STD) MAXLEN(256) +
                           FORCE(*YES) SEQ(*FIFO) SENDERID(*YES) +
                           TEXT('Požadavková fronta (FIFO)')
             ENDDO

/*   Existuje fronta VYSFR v *LIBL?                                    */
             CHKOBJ     OBJ(*LIBL/VYSFRQ) OBJTYPE(*DTAQ)
/*   Jestliže neexistuje, vytvořit ji v *CURLIB                        */
             MONMSG     MSGID(CPF9801) EXEC(DO)
CRTDTAQ    DTAQ(*CURLIB/VYSFR) TYPE(*STD) MAXLEN(256) +
                           FORCE(*YES) SEQ(*KEYED) KEYLEN(36) +
SENDERID(*NO) TEXT('Výsledková fronta +
                           (Keyed)')
             ENDDO
```

Parametr SENDERID (identifikace zasilatele) určuje, zda se má ke zprávě připojit identifikace úlohy, která posílá zprávu do dané fronty.

Identifikace je 44bajtová struktura, kde první dvě čtyřbajtová pole představují pakovaný údaj délky a oba mají hodnotu 44. Další údaje jsou jméno úlohy (10), jméno uživatele (10), číslo úlohy (6) a jméno současného uživatele (10); celkem délky 36 znaků. Jméno současného uživatele je stejné jako jméno prvního uživatele a získá se příkazem RTVJOBA z parametru CURUSER. *(Není jasné, proč se uživatel v identifikaci vyskytuje dvakrát.)* Následující tabulka ukazuje strukturu informací o identifikaci zasilatele (Sender ID).

Offset			
Dec	Hex	Type	Field
0	0	PACKED(7,0)	Bytes returned
4	4	PACKED(7,0)	Bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User profile name
28	1C	CHAR(6)	Job number
34	22	CHAR(10)	Senders current user profile name.

Tvar a obsah záznamů ve frontách si určujeme sami, většinou pomocí datových struktur. Při vytváření objektu datové fronty však uvádíme pouze maximální délku záznamu (zde 256 bajtů). Skutečnou délku záznamu zadáme až při zápisu záznamu do fronty, a zjistíme ji při přečtení záznamu z fronty.

Struktura požadavkové fronty typu FIFO

```
 ************************************************************
 *   Soubor POZAD - Struktura požadavkové zprávy pro frontu POZFR
 ************************************************************
A                                    REF(REF)
A          R POZADR
 *   Řídicí znak
A            ZNAK      R           TEXT('Řídicí znak: E-konec, +
A                                  ostatní-bez významu')
 *   Číslo zboží
A            ZBOZI     R           TEXT('Číslo zboží')
A                                  REFFLD(CZBOZI)
```

```
 *   Identifikace úlohy pro frontu
A            JOBID     R               TEXT('Identifikace úlohy')
A                                      REFFLD(JOBID)
```

Struktura výsledkové fronty typu KEYED

Pro zprávu ve frontě VYSFR se používá struktura datového souboru CENY.

```
 ************************************************************
 *   Soubor CENY - Ceník zboží
 ************************************************************
A                                      UNIQUE
A                                      REF(REF)
A          R CENYR
 *   Číslo zboží
A            CZBOZI    R
 *   Cena za jednotku (kus)
A            CENAJ     R
 *   Název zboží
A            NAZZBO    R
 *   Definice klíče - Číslo zboží
A          K CZBOZI
```

Parametry volání

Parametry zápisové funkce QSNDDTAQ

Required Parameter Group:

1	Data queue name	Input	Char(10)
2	Library name	Input	Char(10)
3	Length of data	Input	Packed(5,0)
4	Data	Input	Char(*)

Optional Parameter Group 1:

5	Length of key data	Input	Packed(3,0)
6	Key data	Input	Char(*)

Optional Parameter Group 2:

7	Asynchronous request	Input	Char(10)

Optional Parameter Group 3:

8	Data is from a journal entry	Input	Char(10)

Parametry čtecí funkce QRCVDTAQ

Required Parameter Group:

1	Data queue name	Input	Char(10)
2	Library name	Input	Char(10)
3	Length of data	Output	Packed(5,0)
4	Data	Output	Char(*)
5	Wait time	Input	Packed(5,0)

Optional Parameter Group 1:

6	Key order	Input	Char(2)
7	Length of key data	Input	Packed(3,0)
8	Key data	I/O	Char(*)
9	Length of sender information	Input	Packed(3,0)
10	Sender information	Output	Char(*)

Optional Parameter Group 2:

11	Remove message	Input	Char(10)
12	Size of data receiver	Input	Packed(5,0)
13	Error code	I/O	Char(*)

Význam a hodnoty parametrů

První čtyři parametry jsou jasné - jméno datové fronty, jméno její knihovny, skutečná délka zapisovaných či přečtených dat v počtu bajtů, data určená k zápisu či data přečtená.

Parametr *Wait time* určuje čas ve vteřinách, po který čtecí funkce čeká, dokud ve frontě není zpráva k přečtení. Hodnota *-1* znamená neomezené čekání, hodnota *0* znamená, že funkce nečeká vůbec.

Nepovinné parametry týkající se klíče jsou jeho data a délka. U čtecí funkce je ještě parametr *Key order*, který určuje, podle jakého kritéria chceme záznam přečíst:

GT - Greater than (větší než)
LT - Less than (menší než)
NE - Not equal (nerovno)
EQ - Equal (rovno)
GE - Greater than or equal (větší nebo rovno)
LE - Less than or equal (menší nebo rovno)

Při tom se porovnávají data klíče z parametru s hodnotou klíče zprávy ve frontě.

Parametr *Sender information* jsme již probrali. Informace o zasilateli se při zápisu připojuje automaticky nebo vůbec, podle zadání při vytvoření fronty.

Další nepovinné parametry v příkladu nepoužíváme.

Program KLIENTR

```
**free

//*************************************************************
//   KLIENTR  Klient s dvěma datovými frontami (FIFO a Keyed).
//            Program nejprve zobrazí výzvu k zadání požadavku
//              na obrazovce. Uživatel vyplní číslo zboží
//              a program toto číslo pošle Serveru jako součást
//              požadavkové zprávy (přes frontu POZFR).
//            Pak čeká na odpověď (výsledek) od Serveru (přes
```

```
//              frontu VYSFR) a pak ji zobrazí na obrazovce.
//*****************************************************************
Ctl-Opt;
//=================================================================
//   Popis souborů
//=================================================================

//   Obrazovkový soubor
Dcl-F CENYW2         WORKSTN ;

//=================================================================
//   Popis dat
//=================================================================

//   Datová struktura požadavkové zprávy pro frontu POZFR
Dcl-DS pozad_ds           extname('POZAD') qualified;
End-DS;

//   Datová struktura výsledkové zprávy pro frontu VYSFR
//     členěná podle souboru CENY
Dcl-DS vysl_ds            extname('CENY') qualified;
End-DS;

//   Parametry pro datové fronty
Dcl-S jm_fronty             Char(10);
Dcl-S knihovna              Char(10) inz('*LIBL');
Dcl-S delka_dat           Packed(5:0);
Dcl-S delka_klice         Packed(3:0);
Dcl-S klic                  Char(36);
Dcl-S cekat               Packed(5:0);
Dcl-S relace                Char(2) inz('EQ');
Dcl-S delka_id            Packed(3:0);
Dcl-S id                    Char(44);
Dcl-S jobid                 Char(36);

// Datová struktura pro výsledky volání API QUSRJOBI
/COPY QSYSINC/QRPGLESRC,QUSRJOBI

// Prototyp API QUSRJOBI
Dcl-PR  usrjobi extpgm('QUSRJOBI'); // Program se nemůže jmenovat stejně jako API,
                                    // protože je tak pojmenovaná konstanta v datové struktuře
   QUSI030000               LikeDS(QUSI030000); // Odvolává se na předchozí strukturu
   del_bin                   Int(10:0);
   format                   Char(8);
   jobname                  Char(26);
   int_job_id               Char(16);
End-PR  usrjobi;

//  Parametry pro API QUSRJOBI
Dcl-S del_bin                Int(10:0) inz(%size(QUSI030000));
Dcl-S format                Char(8) inz('JOBI0300');
Dcl-S jobname               Char(26) inz('*');
Dcl-S int_job_id            Char(16) inz(' ');

// Prototyp API QSNDDTAQ
Dcl-PR QSNDDTAQ extpgm('QSNDDTAQ');
   jm_fronty                Char(10);
   knihovna                 Char(10);
   delka_dat                Packed(5: 0);
   pozad_ds                 LikeDS(pozad_ds);
   delka_klice              Packed(3: 0);
   klic                     Char(36);
End-PR QSNDDTAQ;

// Prototyp API QRCVDTAQ
Dcl-PR QRCVDTAQ extpgm('QRCVDTAQ');
   jm_fronty                Char(10);
   knihovna                 Char(10);
   delka_dat                Packed(5: 0);
   vysl_ds                  LikeDS(vysl_ds);
   cekat                    Packed(5: 0);
   relace                   Char(2);
```

```
   delka_klice              Packed(3: 0);
   jobid                    Char(36);
   delka_id                 Packed(3: 0);
   id                       Char(44);
End-PR QRCVDTAQ;

//=====================================================================
//   Hlavní program
//=====================================================================

//   Nekonečná smyčka
DoW 0 = 0;
   //   Zobrazit zadání čísla zboží
   Exfmt CENYW00;
   If *in03;
      *inlr = *on;
      return;
   EndIf;
   //   Enter:
   //   Získat údaje ze systému (job attributes) k sestavení
   //     identifikace úlohy do hledacího klíče
   //     (V CL by stačil příkaz RTVJOBA)
   Exsr jobatt;

   //   Poslat zprávu Serveru do pozad_ds
   Exsr sndq;

   //   Číst zprávu od Serveru z vysl_ds
   Exsr rcvq;
   //   Když zpráva nepřijde do 10 vteřin - Zapnout ind. 80
   //     pro chybovou zprávu a dosadit otazníky a nuly
   If delka_dat = 0;
      *in80 = *on;
      CZBOZI = *all'?';
      NAZZBO = *all'?';
      CENAJ = 0;
   Else;
      *in80 = *off;
      CZBOZI = vysl_ds.CZBOZI;
      NAZZBO = vysl_ds.NAZZBO;
      CENAJ  = vysl_ds.CENAJ;
   EndIf;

   //   Zobrazit přijatá data zprávy na obrazovce
   Exfmt CENYW01;
   If *in03;
      *inlr = *on;
      return;
   EndIf;

EndDo;  // konec nekonečné smyčky

//   Podprogramy
//=====================================================================
//---------------------------------------------------------------------
//   sndq - Poslat požadavek do fronty POZFR (send to queue)
//---------------------------------------------------------------------
BegSr sndq;
   delka_klice = 0;
   jm_fronty = 'POZFR';
   delka_dat = %len(pozad_ds);
   // Číslo zboží z obrazovky do požadavkové datové struktury
   pozad_ds.ZBOZI = ZBOZI;
   callp QSNDDTAQ ( jm_fronty:
                    knihovna:
                    delka_dat:
                    pozad_ds:      // data POSÍLANÁ
                    delka_klice:    // = 0
                    klic        );  // klíč nepoužit
EndSr;
//---------------------------------------------------------------------
//   rcvq - Číst požadavek z fronty (receive from queue)
```

```
//-----------------------------------------------------------------
BegSr rcvq;
   delka_klice = 36;
   delka_id = 44;
   jm_fronty = 'VYSFR';
   cekat = 10;
   delka_dat = %len(vysl_ds);
   callp QRCVDTAQ ( jm_fronty:
                    knihovna:
                    delka_dat:
                    vysl_ds:       // data PŘIJÍMANÁ
                    cekat:         // čekání v sekundách
                    relace:        // relační operátor
                    delka_klice:
                    pozad_ds.JOBID: // klíč
                    delka_id:
                    id          );
EndSr;
//-----------------------------------------------------------------
//   jobatt - Získat údaje ze systému pro sestavení
//            identifikace úlohy
//-----------------------------------------------------------------
//   API QUSRJOBI pro získání údajů o úloze
//     Výsledek se uloží do datové struktury QUSI030000
BegSr jobatt;
   callp usrjobi ( QUSI030000:    // Výsledek
                   del_bin:       // Délka výsledku
                   format:        // Jménoformátu
                   jobname:       // Jméno úlohy = *
                   int_job_id );  // Interní ID = ' '

   //   Sestavit identifikaci úlohy pro klíč zprávy
   //   = jméno úlohy + jméno uživatele
   //    + číslo úlohy + jméno uživatele ještě jednou.
   pozad_ds.JOBID = QUSJN04 + QUSUN04 + QUSJNBR04 + QUSUN04;
EndSr;
```

Obrazovkový soubor CENYW2 pro server

```
     A*****************************************************************
     A*   Soubor CENYW2 - Obrazovkový soubor pro KLIENTC, KLIENTR      *
     A*****************************************************************
     A                                      DSPSIZ(24 80 *DS3)
     A                                      REF(*LIBL/REF)
     A                                      CA03(03 'End')
      *   Formát pro zadání požadavkové zprávy pro datovou frontu POZFR
     A          R CENYW00
     A                                    2 35'Zadejte číslo zboží'
     A                                    5  2'Číslo zboží....:'
     A            ZNAK      R        H
     A            ZBOZI     R        B  5 20 REFFLD(CZBOZI)
     A                                  23  2'F3=Konec'

      *   Formát pro zobrazování ceníkových údajů
     A          R CENYW01
     A                                    2 35'Ceniková data'
     A                                    2 62DATE EDTCDE(Y)
     A                                    2 72TIME EDTWRD('  -  -  ')
     A                                    5  2'Číslo zboží....:'
     A            CZBOZI    R        O  5 20
     A                                    6  2'Cena...........:'
     A            CENAJ     R        O  6 20 EDTCDE(K)
     A                                    7  2'Název zboží....:'
     A            NAZZBO    R        O  7 20
     A                                  23  2'F3=Konec'
     A  80                              24  2'Server neodpovídá'
     A                                      DSPATR(HI)
     A          R CENYW02
     A                                      WINDOW(*DFT 22 65)
     A                                    2  2'Toto je původní zpráva o zamčené v-
     A                                      ětě:'
```

```
     A                                      DSPATR(HI)
     A            CZPR           7A  O  3  3
     A            ZPRAVA        50A  O  3 11
     A                                   4  2'Tak by to mohlo být česky:'
     A                                      DSPATR(HI)
     A            CESKY          58A  O  5  3
     A            CESKY2         58A  O  6  3
     A                                   7  3'Zkuste zjistit, kdo to je, a troch-
     A                                      u ho/ji popohnat.'
     A                                   8  2'Toto je sekundární text zprávy (he-
     A                                      lp)'
     A                                      DSPATR(HI)
     A            MSG1          240A  B  9  3CNTFLD(060) DSPATR(PR)
     A            MSG2          240A  B 13  3CNTFLD(060) DSPATR(PR)
     A            MSG3          240A  B 17  3CNTFLD(060) DSPATR(PR)
```

Program SERVERR

```
**free
//*****************************************************************
//   SERVERR - Server s dvěma datovými frontami (FIFO a Keyed).
//             Program lze spustit příkazem SBMJOB nebo CALL.
//             Server přijímá požadavky od úloh ve frontě POZFR.
//               V požadavku (zprávě) je řídicí znak a číslo zboží.
//               Řídicí znak E ukládá Serveru ukončit se. Řídicí
//               znak mezera (nebo jiný znak) nemá žádný význam.
//             Podle čísla zboží SERVER najde v databázi záznam
//               a pošle data záznamu zpět úloze, od níž dostal
//               příkaz.
//*****************************************************************

//=================================================================
//   Popis souborů
//=================================================================
Dcl-F CENY                   Keyed;

//=================================================================
//   Popis dat
//=================================================================
//   Datová struktura požadavkové zprávy pro frontu POZFR
Dcl-DS pozad_ds           extname('POZAD') qualified;
End-DS;

//   Datová struktura výsledkové zprávy pro frontu VYSFR
Dcl-DS vysl_ds            likerec(CENYR); // z definice souboru CENY

Dcl-S jm_fronty             Char(10);
Dcl-S knihovna              Char(10) inz('*LIBL');
Dcl-S delka_dat           Packed(5:0);
Dcl-S cekat               Packed(5:0) inz(-1);
Dcl-S relace                Char(2) inz('EQ');
Dcl-S klic                  Char(36);
Dcl-S delka_klice         Packed(3:0) inz(0);
Dcl-S id                    Char(44);
Dcl-S delka_id            Packed(3:0) inz(44);

// Prototyp API QCLRDTAQ
Dcl-PR QCLRDTAQ extpgm('QCLRDTAQ');
   jm_fronty   Char(10);
   knihovna    Char(10);
End-PR QCLRDTAQ;

// Prototyp API QSNDDTAQ
Dcl-PR QSNDDTAQ extpgm('QSNDDTAQ');
   jm_fronty   Char(10);
   knihovna    Char(10);
   delka_dat   Packed(5: 0);
   vysl_ds     LikeDS(vysl_ds);
   delka_klice Packed(3: 0);
   klic        Char(36);
```

```
End-PR QSNDDTAQ;

// Prototyp API QRCVDTAQ
Dcl-PR QRCVDTAQ extpgm('QRCVDTAQ');
   jm_fronty   Char(10);
   knihovna    Char(10);
   delka_dat   Packed(5: 0);
   pozad_ds    LikeDS(pozad_ds);
   cekat       Packed(5: 0);
   relace      Char(2);
   delka_klice Packed(3: 0);
   klic        Char(36);
   delka_id    Packed(3: 0);
   id          Char(44);
End-PR QRCVDTAQ;

//=====================================================================
//   Hlavní program
//=====================================================================
//   Vyčistit obě fronty
jm_fronty = 'POZFR';
callp QCLRDTAQ ( jm_fronty: knihovna  );
jm_fronty = 'VYSFR';
callp QCLRDTAQ ( jm_fronty: knihovna  );

//   Nekonečná smyčka
DoW 0 = 0;
   //   .Číst požadavek z požadavkové fronty (čekání není omezeno)
   Exsr rcvq;
   //   .Je-li řídicí znak E - Končit program (a tedy i úlohu)
   If pozad_ds.ZNAK = 'E';
      *InLR = *on;
      return;
   EndIf;

   //   .Číst záznam z ceníku podle klíče
   Chain pozad_ds.ZBOZI  CENY  vysl_ds;

   //   .Když se nenašel, dosadit nuly a otazníky
   If not %found;
      vysl_ds.CZBOZI = *all'?';
      vysl_ds.NAZZBO = *all'?';
      vysl_ds.CENAJ = 0;
   EndIf;  // (*IN90)
   //   .Zapsat odpověď do výsledkové fronty
   Exsr sndq;
EndDo;  // nekonečná smyčka

//=====================================================================
//   P O D P R O G R A M Y
//=====================================================================
//---------------------------------------------------------------------
//   rcvq - Číst požadavek z fronty (receive from queue)
//          přes datovou strukturu pozad_ds
//---------------------------------------------------------------------
BegSr rcvq;
   jm_fronty = 'POZFR';
   delka_klice = 0;
   delka_id = 44;
   delka_dat = %len(pozad_ds);
   callp QRCVDTAQ ( jm_fronty:
                    knihovna:
                    delka_dat:
                    pozad_ds:     // data PŘIJÍMANÁ
                    cekat:         // čekání v sekundách
                    relace:        // relační operátor
                    delka_klice:   // = 0
                    klic:          // klíč nepoužit
                    delka_id:      // 44
                    id          );
EndSr;
//---------------------------------------------------------------------
```

```
//   sndq - Poslat odpověď do výsledkové fronty (send to queue)
//          přes proměnnou vysl_ds (datová struktura jako CENY)
//-------------------------------------------------------------------
BegSr sndq;
   //   Přesunout proměnné z ceníku do odpovědi pro uživatele
   //   36 znaků od 9. pozice proměnné ID tvoří klíč JOBID
   klic = %subst(id: 9: 36);
   //   Poslat zprávu do výsledkové fronty
   jm_fronty = 'VYSFR';
   delka_klice = 36;
   delka_dat = %len(vysl_ds);
   callp QSNDDTAQ ( jm_fronty:
                    knihovna:
                    delka_dat:
                    vysl_ds:      // data POSÍLANÁ
                    delka_klice:
                    klic        );
EndSr;
```

Definice a prototypy k procedurám typu UNIX

Definice a prototypy jsou zapsány v pevném sloupcovém formátu RPG IV. V programech s volným formátem příkazů je lze použít jen prostřednictvím direktivy /COPY nebo /INCLUDE. Lze je přepsat také do volného tvaru, viz tento příklad.

```
     ****************************************************************
     *
     *   Prototypy podprocedur pro sokety a potrebne popisy dat
     *
     ****************************************************************
     *-- Socket address information structure ------------------------

    D SocketAddr      DS
    D   SinFamily                    5I 0
    D   SinPort                      5U 0
    D   SinAddr                     10U 0
    D   SinZero                      8A   Inz( *ALLX'00' )
    D SockAddr        S                *   Inz( %Addr(SocketAddr) )
    D AddressLength   S              10I 0
    D AddrLen         S                *   Inz( %Addr(AddressLength) )


     *-- Internet address structure ----------------------------------

    D InAddr          DS
    D   BinAddr                     10U 0

    D InAddrLen       S              10I 0 Inz( %Size(InAddr) )
    D InAddrP         S                *   Inz( %Addr(InAddr) )


     *-- Host entry returned pointers and data -----------------------

    D HostEnt         DS                   Align Based(HostEntP)
    D  HNameP                        *
    D  HAliasesP                     *
    D  HAddrType                    10I 0
    D  HLength                      10I 0
    D  HAddrListP                    *

    D  HAddrList      S                *   Based(HAddrListP) Dim(100)
    D  HAddr          S              10U 0 Based(HAddrP)
    D  HAddrP         S                *


     *-- I/O options (Fcntl) -----------------------------------------

    D F_SETFL         S              10I 0 Inz(7)
    D O_NONBLOCK      S              10I 0 Inz(128)

    D EWOULDBLOCK     C                    3406


     *-- Error related data ------------------------------------------

    D ErrNo           S              10I 0 Based(ErrNoP)
    D ErrNoP          S                *   Inz
    D ErrMsg          S              60A   Based(ErrMsgP)
    D ErrMsgP         S                *   Inz


     *-- Address families --------------------------------------------

    D AF_UNIX         C                    1
    D AF_INET         C                    2
    D AF_NS           C                    6
    D AF_TELEPHONY    C                    99


     *-- Socket types ------------------------------------------------

    D SOCK_STREAM     C                    1
    D SOCK_DGRAM      C                    2
    D SOCK_RAW        C                    3
    D SOCK_SEQPACKET  C                    5

    D SOL_SOCKET      C                    -1


     *-- Socket level options ----------------------------------------

    D SO_BROADCAST    C                    5
    D SO_DEBUG        C                    10
    D SO_DONTROUTE    C                    15
    D SO_ERROR        C                    20
    D SO_KEEPALIVE    C                    25
    D SO_LINGER       C                    30
```

```
D SO_REUSEADDR    C                   55


 *-- Internet address specifications ------------------------------

D INADDR_ANY      C                   0
D INADDR_BROADCA  C                   -1
D INADDR_LOOPBCK  C                   X'7F000000'
D INADDR_NONE     C                   -1


 ****************************************************************
 *
 *   Konstanty pro Open Stream File
 *
 ****************************************************************

 *** member QSYSINC/H.FCNTL ***********************************

 *-- File Access Modes --------------------------------------------
 *#define O_RDONLY   00001        /* Open for reading only           */
D O_RDONLY         S             10I 0 Inz(X'01')
 *#define O_WRONLY   00002        /* Open for writing only           */
D O_WRONLY         S             10I 0 Inz(x'02')
 *#define O_RDWR     00004        /* Open for reading and writing    */
D O_RDWR           S             10I 0 Inz(x'04')
 *#define O_CREAT      00010      /* Create file if it doesn't exist */
D O_CREATE         S             10I 0 Inz(x'08')
 *#define O_EXCL       00020      /* Exclusive use flag              */
D O_EXCL           S             10I 0 Inz(x'10')
 */*                   00040         reserved                      */
 *#define O_TRUNC      00100      /* Truncate flag                   */
D O_TRUNC          S             10I 0 INZ(x'40')

 *-- File Status Flags --------------------------------------------
 *#define O_CODEPAGE   040000000  /* code page flag                  */
D O_CODEPAGE       S             10I 0 INZ(x'800000')
 *#define O_TEXTDATA  0100000000  /* text data flag                  */
D O_TEXTDATA       S             10I 0 INZ(x'01000000')
 *#define O_APPEND     00400      /* Set append mode                 */
D O_APPEND         S             10I 0 INZ(x'0100')
 *#define O_LARGEFILE  004000000000  /* Large file access            */
D O_LARGEFILE      S             10I 0 INZ(x'20000000')
 *#define O_INHERITMODE 001000000000 /* inherit mode flag            */
D O_INHERITMODE    S             10I 0 INZ(x'08000000')

 *-- Share Mode Values --------------------------------------------
 *#define O_SHARE_RDONLY 000000200000 /* Share with readers only     */
D O_SHARE_RDONLY   S             10I 0 INZ(x'010000')
 *#define O_SHARE_WRONLY 000000400000 /* Share with writers only     */
D O_SHARE_WRONLY   S             10I 0 INZ(x'020000')
 *#define O_SHARE_RDWR   000001000000 /* Share with readers and writers */
D O_SHARE_RDWR     S             10I 0 INZ(x'040000')
 *#define O_SHARE_NONE   000002000000 /* Share with neither readers
 *                                       nor writers                */
D O_SHARE_NONE     S             10I 0 INZ(x'080000')

 *** member QSYSINC/SYS.STAT **********************************

 *-- Definitions of File Modes and File Types ---------------------     */
 *   #define S_IRUSR 0000400    /* Read for owner                     */
D S_IRUSR          S             10I 0 INZ(x'0100')
 *   #define S_IWUSR 0000200    /* Write for owner                    */
D S_IWUSR          S             10I 0 INZ(x'80')
 *   #define S_IXUSR 0000100    /* Execute and Search for owner       */
D S_IXUSR          S             10I 0 INZ(x'40')
 *   #define S_IRWXU (S_IRUSR|S_IWUSR|S_IXUSR)  /* Read, Write,
 *                                                  Execute for owner  */
D S_IRWXU          S             10I 0 INZ(x'01C0')

 *   #define S_IRGRP 0000040         /* Read for group                */
D S_IRGRP          S             10I 0 INZ(x'20')
 *   #define S_IWGRP 0000020         /* Write for group               */
D S_IWGRP          S             10I 0 INZ(x'10')
 *   #define S_IXGRP 0000010         /* Execute and Search for group  */
D S_IXGRP          S             10I 0 INZ(x'08')
 *   #define S_IRWXG (S_IRGRP|S_IWGRP|S_IXGRP)  /* Read, Write,
 *                                                  Execute for group  */
D S_IRWXG          S             10I 0 INZ(x'38')

 *   #define S_IROTH 0000004         /* Read for other                */
D S_IROTH          S             10I 0 INZ(x'04')
 *   #define S_IWOTH 0000002         /* Write for other               */
D S_IWOTH          S             10I 0 INZ(x'02')
 *   #define S_IXOTH 0000001         /* Execute and Search for other  */
D S_IXOTH          S             10I 0 INZ(x'01')
 *   #define S_IRWXO (S_IROTH|S_IWOTH|S_IXOTH)  /* Read, Write,
 *                                                  Execute for other  */
D S_IRWXO          S             10I 0 INZ(x'07')
```

```
 ****************************************************************
 *
 *   Prototypy podprocedur
 *
 ****************************************************************

 *-- Socket --- Create a socket ---------------------------------

 *   int socket(int address_family,
 *              int type,
 *              int protocol)

D Socket          Pr            10I 0 Extproc('socket')
D  AddrFamily                   10I 0 Value
D  SocketType                   10I 0 Value
D  Protocol                     10I 0 Value


 *-- Setsockopt --- Set socket options

 *   int setsockopt(int socket_descriptor,
 *                  int level,
 *                  int option_name,
 *                  char *option_value,
 *                  int option_length)
 *
D SetsockOpt      Pr            10I 0 Extproc('setsockopt')
D  SocketDescr                  10I 0 Value
D  Level                        10I 0 Value
D  OptionName                   10I 0 Value
D  OptionValueP                   *   Value
D  OptionLength                 10I 0 Value


 *-- Bind --- Bind to a socket ----------------------------------

 *   int bind(int socket_descriptor,
 *            struct sockaddr *local_address,
 *            int address_length)

D Bind            Pr            10I 0 ExtProc('bind')
D  SocketDescr                  10I 0 Value
D  LocalAddrP                     *   Value
D  AddrLength                   10I 0 Value


 *-- Listen --- Invite for the incoming connections requests

 *   int listen(int socket_descriptor,
 *              int back_log);

D Listen          Pr            10I 0 ExtProc('listen')
D  SocketDescr                  10I 0 Value
D  BackLog                      10I 0 Value


 *-- Accept --- Accept an incoming connections request

 *   int accept(int socket_descriptor,
 *              struct sockaddr *address,
 *              int *address_length);

D Accept          Pr            10I 0 ExtProc('accept')
D  SocketDescr                  10I 0 Value
D  SocketAddrP                    *   Value
D  AddrLengthP                    *   Value


 *-- InetAddr --- Transform IP address from dotted form ----------

 *   unsigned long inet_addr(char *address_string);

D InetAddr        Pr            10U 0 ExtProc('inet_addr')
D  AddrStringP                    *   Value


 *-- GetHostByName --- Get host address from name ----------------

 *   struct HostEnt {
 *      char   *h_name;
 *      char   **h_aliases;
 *      int    h_addrtype;
 *      int    h_length;
 *      char   **h_addr_list;
 *   };

 *   struct HostEnt *GetHostByName(char *host_name);

D GetHostByName   Pr              *   Extproc('gethostbyname')
D  HostNameP                      *   Value
```

```
 *-- Connect --- Connect to the server

 *   int connect(int socket_descriptor,
 *               struct sockaddr *destination_address,
 *               int address_length);

D Connect         Pr            10I 0 Extproc('connect')
D  SocketDescr                  10I 0 Value
D  DestinAddrP                    *   Value
D  AddrLength                   10I 0 Value


 *-- Read --- Read data from the socket

 *   ssize_t read(int descriptor,
 *                void *buffer,
 *                size_t buffer_length);

D Read            Pr            10I 0 Extproc('read')
D  Descriptor                   10I 0 Value
D  BufferP                        *   Value
D  BufferLength                 10U 0 Value


 *-- Write --- Write data to the socket

 *   ssize_t write(int file_descriptor,
 *                 const void *buffer,
 *                 size_t buffer_length);

D Write           Pr            10I 0 ExtProc('write')
D  Descriptor                   10I 0 Value
D  BufferP                        *   Value
D  BufferLength                 10U 0 Value


 *-- Close --- Close a socket

 *   int close(int descriptor)

D Close           Pr            10I 0 ExtProc('close')
D  Descriptor                   10I 0 Value


 *-- GetErrNo ---- Get error number ----------------------------------

 *   extern int * __errno(void);

D GetErrNo        Pr              *   ExtProc('__errno')


 *-- StrError ---- Get error text ------------------------------------

 *   char *strerror(int errnum);

D StrError        Pr              *   ExtProc('strerror')
D  ErrNumber                    10I 0 Value


 *-- Open -------- Open Stream File ----------------------------------

 *  int open(const char *path, int oflag, . . .);

D Open            Pr            10I 0 ExtProc('open')
D  PathP                          *   Value Options(*String)
D  Oflag                        10I 0 Value
D  Mode                         10U 0 Value Options(*Nopass)
D  Codepage                     10U 0 Value Options(*Nopass)


 *-- OpenDir ------ Open Directory -----------------------------------

 *   DIR *opendir(const char *dirname);

D OpenDir         Pr              *   ExtProc('opendir')                Pointer to DIR
D  DirNameP                       *   Value                             Pointer to dirname


 *-- ReadDir ------ Read directory entry -----------------------------

 *   struct dirent *readdir(unsigned int *DIR);

D ReadDir         Pr              *   ExtProc('readdir')                Pointer to DirEnt
D  DIRP                           *   Value                             Pointer to DIR


 *-- CloseDir ----- Close directory ----------------------------------

 *   int closedir(unsigned int *DIR);
```

```
D CloseDir        Pr              *   ExtProc('closedir')               Pointer to int
D  DIRP                           *   Value                             Pointer to DIR


 *-- Rename ------- Rename directory ---------------------------------

 *   int rename(const char *old, const char *new);

D Rename          Pr            10I 0 ExtProc('Qp0lRenameKeep')         Return code
D  OldNameP                       *   Value                             Pointer to oldname
D  NewNameP                       *   Value                             Pointer to newname


 *-- Unlink ------- Delete file --------------------------------------

 *   int unlink(char *path);

D Unlink          Pr            10I 0 ExtProc('unlink')                 Return code
D  PathP                          *   Value                             Pointer to Path


 *-- RmDir -------- Remove directory ---------------------------------

 *   int rmdir(char *path);

D RmDir           Pr            10I 0 ExtProc('rmdir')                  Return code
D  PathP                          *   Value                             Pointer to Path


 *-- MkDir -------- Make directory -----------------------------------

 *   int mkdir(const char *path, mode_t mode);

D MkDir           Pr            10I 0 ExtProc('mkdir')                  Return code
D  PathP                          *   Value                             Pointer to Path
D  Mode                       10U 0 Value                               Permission bits
```