e-business

Object REXX

Praktische Verwendung von REXX zur Systemverwaltung und Administrierung

Gunter Thum (thu@de.ibm.com)

REXX / Object REXX Entwicklung

IBM Deutschland Entwicklung

FFG der GUUG in Bochum
26. - 28. Maerz 2003, Bochum

REXX

IBM

e-business

Agenda

- Einführung
  - Was ist REXX?
- Ziel des Vortrages
- Vorteile
  - Mächtiger Funktionsumfang
  - Unterstuetzt prozedurale wie auch object- orientierte Programmierung
  - C-API
  - ...
- Eigenschaften
  - Freies Format
  - Einfache Syntax
  - Keine Datentyp-Deklaration
- Praktische Anwendungen im Bereich Systemverwaltung und Administrierung
- Hinweise

REXX

IBM

Was ist REXX?

- REXX ist eine Programmiersprache, die
  - leicht zu erlernen
  - einfach anzuwenden
  - sehr mächtig

  ist
- Rexx ist verfuegbar auf
  - Linux/Intel, Linux PPC und Linux z/OS
  - AIX
  - Sun/Solaris
  - OS/2
  - Windows
  - zOS, zVM, VSE
- Rexx gibt es in 2 Ausprägungen:
  - Classic REXX (zOS/zVM/VSE und OS/2)
  - Object REXX (Linux, AIX, Sun/Solaris, OS/2, Windows)
- REXX ist beschrieben im ANSI standard (X3.274-1996)

Ziel des Vortrages

- Grundlagen der Programmiersprache zeigen
- Anhand von einfachen Beispielen die Mächtigkeit von Object REXX demonstrieren
- An konkreten Beispielen (Systemverwaltung und Administrierung) zeigen, wie Object REXX mit Subsystemen interagiert

e-business

Vorteile

- Mächtiger Funktionsumfang zur Bearbeitung von (z.B.)
  - Strings (compare, copies, length, substr....)
  - streams(charin, charout, linein, lineout...)
  - queues (pull, push, queued...)
- Umfangreiche Klassenbibliothek
  - Array, Stream, Stem, Relation, Directory, Monitor, Message...
- Umfangreiche Utility Bibliothek
  - Manipulation von files und directories, semaphores, macros
- Mehrere Pakete werden standardmäsig mit ausgeliefert
  - mathematisches Funktionspaket, FTP, Socket, Regulare Expressions

- IPC mittels Named Queues
- Macrospace Interface
- SAA-API(C/C++->REXX, REXX -> C/C++)
- Unterstützung von Concurrency, parsing, Datums-Aufbereitung, benutzerdefinierter beliebiger Genauigkeit bei der Berechnung von mathematische Operationen,u.v.m.

IBM

REXX

e-business

Eigenschaften

Freies Format

Folgende 3 Scripts erzeugen das gleiche Resultat:

```
/*a small loop*/
do 5
  say 'hello world'
end
```

```
/*a small loop*/
do 5;
  say 'hello world';
end;
```

```
/*a small loop*/
do 5;  say 'hello world'; end
```

IBM

REXX

Eigenschaften

– Einfache Syntax (an 3 Beispielen)

```
/*Initialization and buildin function execution*/
string_1 = 'This is string 1'
string_2 = 'This is string 2'
say words(string_1)
string_3 = string_1 || string_2
say  string_3
```

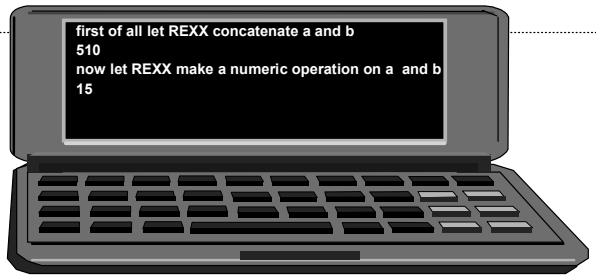
```
/*conditional looping*/
a = 0
do while a < 5
   say 'hello'
   a = a + 1
end
```

```
/*sending a command to the subsystem*/
'ls -ltr '
say rc        /* special variable were language
              /*processor puts return codes
```

Eigenschaften

– Keine Datentypdeklaration

```
/*REXX decides the suitable Datatype*/
a = 5
b = 10
say 'first of all let REXX concatenate a and b'
say  a||b                                    -- a and b are treated as  strings
say 'now let REXX make a numeric operation on a and b'
say  a +b                                    -- a and b are treated as numbers
return
```



Objektorientierte Programmierung

- Neben der prozeduralen Programmierung unterstützt Object REXX die grundlegenden Ideen der objekt-orientierten Programmierung:
  - Erzeugen von Klassen:
    ::CLASS *Klassenname*
  - Erzeugen der Methoden einer Klasse:
    ::METHOD *Methodenname*
  - Erzeugen von Objekten einer Klasse:
    objectname = .Klassenname~new
  - Datenkapselung
  - Vererbung
  - Polymorphie

Eine Object REXX Klasse

```
/* Object / Class creation*/
disk_a = .disk~new('/dev/sda')
say disk_a~size
disk_a~partitions
return
::CLASS disk
::METHOD init
  expose name
  use arg name
  return
::METHOD size
  expose name
  'sfdisk -s ' name ' | rxqueue'
  pull size
  return 'size of disk: ' size
::METHOD partitions
  expose name
  'fdisk -l |grep 'name ' | rxqueue'
  pull disk_name
  do while queued() > 0
    parse pull partition
    parse value partition with partition .
    say 'partition name of disk ' name':'  partition
  end
```

Systemverwaltung und Administrierung
Teil 1: Interaktion mit dem Subsystem

Grundlagen:

- Aufruf von Systemkommands immer in Hochkomma
- Kommunikation zwischen verschiedenen Programm-teilen(threads) durch die Session Queue
- Kommunikation zwischen Prozessen (IPC) durch External Data Queues

Der Filter rxqueue nimmt die Ausgabe anderer Programme und stellt sie in eine REXX queue.
'ls -ltr | rxqueue'

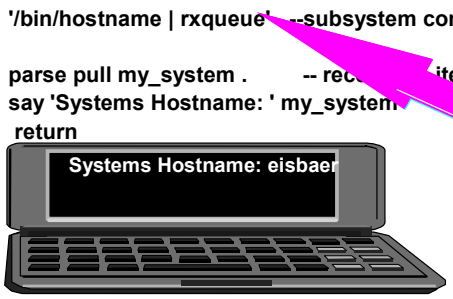
- Das Auslesen der Queue erfolgt (z.B.) mit der Funktion 'pull'

Praktische Beispiele fuer System-verwaltung und Administrierung
(Teil 1: Subsystem-Ausgabe empfangen)

```
/* receiving hostname from system and pipe into REXX queue*/
/* rxqueue receives output and places them on a REXX queue*/

'/bin/hostname | rxqueue'   --subsystem command always in quotes

parse pull my_system .      -- rec      item from the REXX queue
say 'Systems Hostname: ' my_system
return
```



Pipes output from command into the REXX queue

Systemverwaltung und Administrierung
Teil 2: Subsystemausgabe bearbeiten

Grundlagen:

- **mit dem Kommand 'queued()' wird die Anzahl der verbleibenden Einträge in der Daten Queue zurueckgegeben**

Praktische Beispiele fuer System- verwaltung und Administrierung
(Teil 2: Subsystem-Ausgabe bearbeiten)

```
/*create  new userid */
say 'enter userid to create: '
parse pull new_userid
exist_flag = 0
'getent passwd |  rxqueue'          --pipe output from passwd db into REXX queue
do queued()                        -- queued() returns number of entries in queue
  parse pull entry                 -- read next entry from queue
  parse value entry with CreateId ':'  .
  if  CreateId = new_userid then
  do
      exist_flag = 1
      leave
  end
  else iterate
end
if exist_flag = 1 then
  say 'userid ' new_userid ' already exists on the
else do
  'useradd' add_userid
  say 'userid ' new_userid 'created'
end
return
```

Systemverwaltung und Administrierung
Teil 3: Environement variable setzen

Grundlagen:

- **mit der Funktion 'value()' können Environment variable manipuliert werden.**

```
PATH_env = value('PATH',new, 'ENVIRONMENT')
```

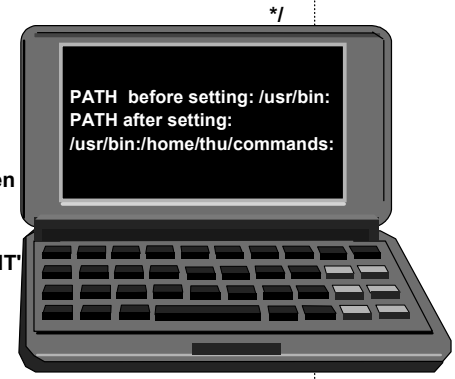
e-business

Praktische Beispiele fuer System- verwaltung und Administrierung (Teil 3: Shell Variable setzen)

```
/* enhance the PATH settings */
/* The Shell variable 'PATH' lists the directories that the shell searches */
/* for commands                                                            */

command_dir = '/home/thu/commands:'
user_path = value('PATH',,'ENVIRONMENT')
say 'PATH before setting: ' user_path

if wordpos(command_dir, user_path) = 0 then
do
  user_path = command_dir || user_path
  call value PATH, user_path, 'ENVIRONMENT'
end
say 'PATH after setting: '
say value('PATH',,'ENVIRONMENT')
return
```



IBM

Systemverwaltung und Administrierung
Teil 4: Systemüberwachung

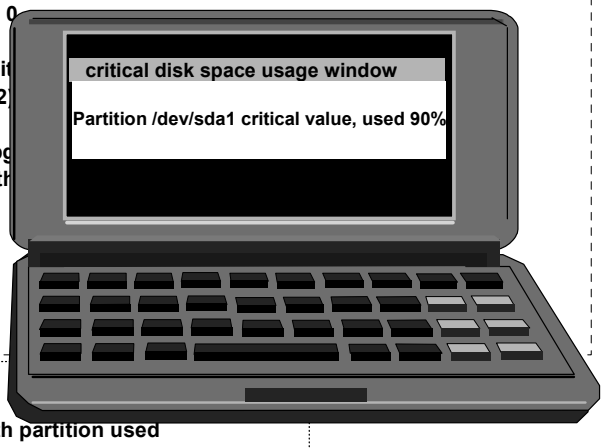
Grundlagen:

- **mit dem Program 'xterm' kann ein neuer REXX-Prozess von einem laufenden REXX-Program gestartet werden. Argumente können 'per Value' oder 'per Referenz' uebergeben werden.**

Die Entgegennahme der Argumente 'per Value' erfolgt mit: parse arg
Die Entgegennahme der Argumente 'per Referenz' erfolgt mit: use arg

Praktische Beispiele fuer System-verwaltung und Administrierung (Teil 4: Systemüberwachung)


```
/* checking disk space usage (start as Background process using f.e. at,
batch*/
do forever
 'df |grep /dev |rxqueue'                --reports amount of free disk space
 do while queued() > 0
   parse pull line
   parse value line wit
   if substr(pused,1,2)
   do
      'xterm -fg red -bg
      ' -e rexx /home/th
   end
 end
 call SysSleep 3600
end
return
```



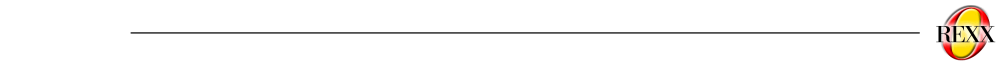

```
/*output window*/
parse arg input
parse value input with partition used

say 'Partition ' partition 'crititcal value, used ' used
pull
return
```

Systemverwaltung und Administrierung
Teil 5: Logon security

Grundlagen:

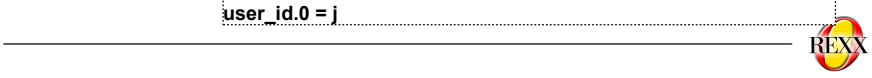
- **Mit Object REXX ist es sehr einfach möglich Dateien auszuwerten**
  - Die stream Klasse bietet Methoden zur Bearbeitung von Dateien
  - Parse bietet eine Vielzahl von Optionen zur Auswertung von Informationen
  - Die Regular Expression Klasse ermöglicht es Informationen mit vorgegebenen Mustern zu vergleichen



Praktische Beispiele fuer System- verwaltung und Administrierung (Teil 5: Logon Security)

```
/* controlling active users on the system */
data. = ''                                  -- initialize REXX stem
password_file = .stream~new('/etc/shadow')
password_file~open('read')
i = 0
do while password_file~lines > 0            -- read all entries from shadow file
   i = i + 1
   data.i = password_file~linein             -- put entries into stem
end
data.0 = i                                  -- remember number of entries
password_file~close
```

```
'finger -l |rxqueue'                        -- give me active users
j = 0
do while queued() > 0
 parse pull line
 if wordpos('Login:',line) > 0 then
 do
  j = j +1
  parse value line with .  user_id.j .    --extract logged on users
 end
end
user_id.0 = j
```



e-business

Praktische Beispiele fuer System- verwaltung und Administrierung (Teil 5: Logon Security)

```
exist_flag = 0
do j = 1 to user_id.0                          --Go through all logged on users
  do i = 1 to data.0                           -- Go through the whole password db
    if pos(strip(user_id.j), data.i) > 0 then
    do
      parse value data.i with user_id ':' password ':'
      if password = '' || password = ' ' then  -- If user logged on an account with no password
      do
        say 'UserId ' user_id 'logged on the system with an account that has no password'
        exist_flag = 1
      end
    end
  end
end
if exist_flag = 0 then
  say ' No critical User logged o
```

IBM

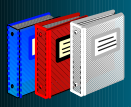
REXX

e-business

Further information

Object REXXhomepage:
http://www.ibm.com/software/ad/obj-rexx

Documentation:
"Object REXX for Windows95/NT with OODialog"
ISBN 0-13-858028-6 (includes CD with Object REXX)



IBM

REXX