GSE GUIDE SHARE EUROPE

NL Region

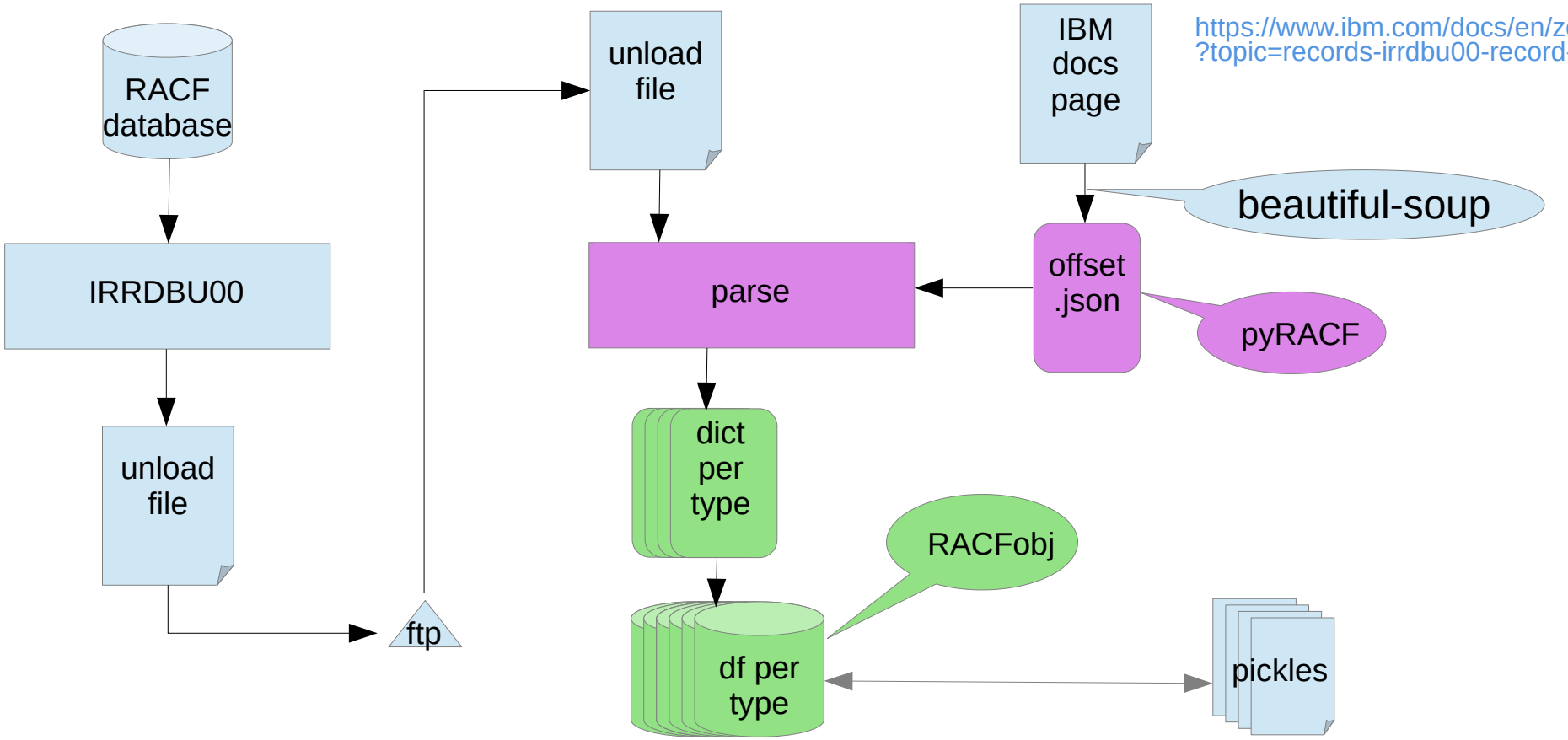
PLATFORM ENGINEERING TO THE FUTURE

pyRACF:
Jurassic Frame meets Pythonic Impact

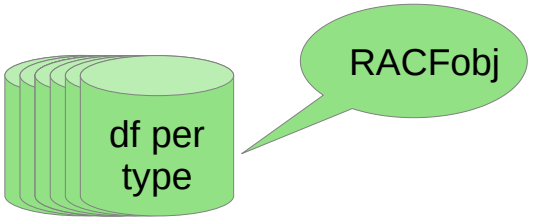
Henri Kuiper

Rob van Hoboken

Frame your RACF profiles



Use those Frames




```
import RACF from pyracf

r = RACF("my.unload")
r.parse() # create DataFrames

r.groups  # all groups
r.users  # all users
```

Type	Prefix	DataFrame property	Description
0100	GPBD	groups	Group Basic Data
0101	GPSGRP	subgroups	Group Subgroups
0102	GPMEM	connects	Group Members
0103	GPINSTD	groupUSRDATA	Group Installation Data
0110	GPDFP	groupDFP	Group DFP Data
0120	GPOMVS	groupOMVS	Group OMVS Data
0130	GPOVM	groupOVM	Group OVM Data
0141	GPTME	groupTME	Group TME Data
0151	GPCSD	groupCSDATA	Group CSDATA custom fields
0200	USBD	users	User Basic Data
0201	USCAT	userCategories	User Categories
0202	USCLA	userClasses	User Classes
0203	USGCON	groupConnect	User Group Connections
0204	USINSTD	userUSRDATA	User Installation Data
0205	USCON	connectData	User Connect Data
0206	USRSF	userRRSFDATA	RRSF data
0207	USCERT	userCERTname	user certificate name
0208	USNMAP	userAssociationMapping	User Associated Mappings
0209	USDMAP	userDistributedIdMapping	User Associated Distributed Mappings
020A	USMFA	userMFAfactor	user Multifactor authentication data

Framed groups...

- Columns with data
  - Names with prefix
    - GPBD
  - From IBM docs
- Ellipsis …
  - Many rows
  - Many columns
- Slow to search

```
[6]: r.groups
```

[6]:

	GPBD_RECORD_TYPE	GPBD_NAME	GPBD_SUPGRP_ID	GPBD_CREATE_DATE	GPBD_OWNER_ID	GPBD_UACC	GPBD_NOTERMUACC	GPBD_INSTALL_DATA	GPBD_M
0	0100	#ACCLIST	SYS1	2012-04-26	SYS1	NONE	NO		
1	0100	#CONNECT	SYS1	2012-04-26	SYS1	NONE	NO		
2	0100	#FULL	SYS1	2012-04-25	SYS1	NONE	NO		
3	0100	#GROUP	SYS1	2012-04-25	SYS1	NONE	NO		
4	0100	#HLPDESK	SYS1	2012-04-25	SYS1	NONE	NO		
...	...	...	...	...	...	...	...	...	
1283	0100	ZOSUGRP	SYS1	2019-11-11	SYS1	NONE	NO	ZOSCONN UNAUTH GROUP	
1284	0100	ZSECURE	SYS1	2014-05-09	SYS1	NONE	NO		
1285	0100	ZWEADMIN	SYS1	2020-10-26	SYS1	NONE	NO	ZOWE ADMIN GROUP	
1286	0100	ZWE100	SYS1	2020-10-26	SYS1	NONE	NO	ZOWE - HLQ STUB	
1287	0100	ZWE200	SYS1	2023-02-18	IBMUSER	NONE	NO	ZOWE - HLQ STUB	

1288 rows × 10 columns

Use pandas methods to search

- List group name starting with SYS
  - r.groups.loc[r.groups.GPBD_NAME.str.match('SYS.*')]
    - Select on (combination) of data fields
  - r.groups.query("GPBD_NAME.str.contains('SYS')")
    - Quotes within quotes
  - Life-long employment for pandas expertise

```
*[9]: r.groups.loc[r.groups.GPBD_NAME.str.match('SYS.*')]
```

[9]:

	_NAME	GPBD_RECORD_TYPE	GPBD_NAME	GPBD_SUPGRP_ID	GPBD_CREATE_DATE	GPBD_OWNER_ID	GPBD_UACC	GPBD_NOTERMUACC	GPBD_INSTALL_DATA
0	SYSAUDIT	0100	SYSAUDIT	ZSECURE	2014-05-09	ZSECURE	NONE	NO	GROUP FOR SYSTEM AUDITORS
1	SYSCSF	0100	SYSCSF	SYS1	2019-11-12	SYS1	NONE	NO	
2	SYSCTLG	0100	SYSCTLG	SYS1	1995-06-06	IBMUSER	NONE	NO	
3	SYSHSM	0100	SYSHSM	SYS1	2023-09-13	SYS1	NONE	NO	
4	SYSPROG	0100	SYSPROG	SYS1	2014-05-09	SYS1	NONE	NO	
5	SYSRACF	0100	SYSRACF	SYS1	2022-09-26	SYS1	NONE	NO	
6	SYS1	0100	SYS1		1995-06-06	IBMUSER	NONE	NO	
7	SYS2	0100	SYS2	SYS1	2023-07-07	SYS1	NONE	NO	

Faster Frames

- pyRACF adds index
  - To all Frames
  - Fast search
  - Consistent correlation
- 1 index field:
  - Group, user ID, data set
- 2 index fields:
  - Connect info (group, user), general resource profiles (class, profile key)
- 3: data set permits, 4: general resource permits

```
[7]: r.groups
```

[7]:

_NAME	GPBD_RECORD_TYPE	GPBD_NAME	GPBD_SUPGRP_ID	GPBD_CREATE_DATE	GPBD_OWNER_ID	GPBD_UACC	GPBD_NOTERMUACC	GPBD_INSTALL_DATA	G
#ACCLIST	0100	#ACCLIST	SYS1	2012-04-26	SYS1	NONE	NO		
#CONNECT	0100	#CONNECT	SYS1	2012-04-26	SYS1	NONE	NO		
#FULL	0100	#FULL	SYS1	2012-04-25	SYS1	NONE	NO		
#GROUP	0100	#GROUP	SYS1	2012-04-25	SYS1	NONE	NO		
#HLPDESK	0100	#HLPDESK	SYS1	2012-04-25	SYS1	NONE	NO		
...	...	...	...	...	...	...	...	...	
ZOSUGRP	0100	ZOSUGRP	SYS1	2019-11-11	SYS1	NONE	NO	ZOSCONN UNAUTH GROUP	
ZSECURE	0100	ZSECURE	SYS1	2014-05-09	SYS1	NONE	NO		
ZWEADMIN	0100	ZWEADMIN	SYS1	2020-10-26	SYS1	NONE	NO	ZOWE ADMIN GROUP	
ZWE100	0100	ZWE100	SYS1	2020-10-26	SYS1	NONE	NO	ZOWE - HLQ STUB	
ZWE200	0100	ZWE200	SYS1	2023-02-18	IBMUSER	NONE	NO	ZOWE - HLQ STUB	

1288 rows × 10 columns

Search Frame

- Plural, all profiles in the Frame
  - r.groups
  - r.users
  - Property
    - so no parm, no parens

```
[7]: r.groups
```

[7]:

_NAME	GPBD_RECORD_TYPE	GPBD_NAME	GPBD_SUPGRP_ID	GPBD_CREATE_DATE	GPBD_OWNER_ID	GPBD_UACC	GPBD_NOTERMUACC	GPBD_INSTALL_DATA
#ACCLIST	0100	#ACCLIST	SYS1	2012-04-26	SYS1	NONE	NO	
#CONNECT	0100	#CONNECT	SYS1	2012-04-26	SYS1	NONE	NO	
#FULL	0100	#FULL	SYS1	2012-04-25	SYS1	NONE	NO	
#GROUP	0100	#GROUP	SYS1	2012-04-25	SYS1	NONE	NO	
#HLPDESK	0100	#HLPDESK	SYS1	2012-04-25	SYS1	NONE	NO	
...	...	...	...	...	...	...	...	...
ZOSUGRP	0100	ZOSUGRP	SYS1	2019-11-11	SYS1	NONE	NO	ZOSCONN UNAUTH GROUP
ZSECURE	0100	ZSECURE	SYS1	2014-05-09	SYS1	NONE	NO	
ZWEADMIN	0100	ZWEADMIN	SYS1	2020-10-26	SYS1	NONE	NO	ZOWE ADMIN GROUP
ZWE100	0100	ZWE100	SYS1	2020-10-26	SYS1	NONE	NO	ZOWE - HLQ STUB
ZWE200	0100	ZWE200	SYS1	2023-02-18	IBMUSER	NONE	NO	ZOWE - HLQ STUB

1288 rows × 10 columns

- Singular, 1 profile
  - r.user('IBMUSER')
  - Function method
    - Quotes needed
  - Empty Frame if no match

```
[11]: r.user('IBMUSER')
```

[11]:

_NAME	USBD_RECORD_TYPE	USBD_NAME	USBD_CREATE_DATE	USBD_OWNER_ID	USBD_ADSP	USBD_SPECIAL	USBD_OPER	USBD_REVOKE	USBD_GRPACC	USBD
IBMUSER	0200	IBMUSER	1995-06-06	IBMUSER	NO	YES	YES	YES	NO	

1 rows × 52 columns

Search Frame, advanced

- Selection methods
  - .find( )
    - index value(s)
      - Generic pattern
      - quotes
    - column headers
      - Strip prefix…
      - Single =
      - Quoted value
  - .skip( )
    - same

```
[14]: r.users.find('IBM*')
```

[14]:

_NAME	USBD_RECORD_TYPE	USBD_NAME	USBD_CREATE_DATE	USBD_OWNER_ID	USBD_ADSP	USBD_SPECIAL	USBD_OPER	USBD_REVOKE	USBD_GRPACC	USB
IBMUSCN	0200	IBMUSCN	2019-11-11	IBMUSER	NO	NO	NO	NO	NO	
IBMUSECN	0200	IBMUSECN	2019-11-11	IBMUSER	NO	NO	NO	NO	NO	
IBMUSER	0200	IBMUSER	1995-06-06	IBMUSER	NO	YES	YES	YES	NO	

3 rows × 52 columns

```
[15]: r.users.find(SPECIAL='YES')
```

[15]:

_NAME	USBD_RECORD_TYPE	USBD_NAME	USBD_CREATE_DATE	USBD_OWNER_ID	USBD_ADSP	USBD_SPECIAL	USBD_OPER	USBD_REVOKE	USBD_GRPACC	USB
ADCDMST	0200	ADCDMST	2004-11-10	SYS1	NO	YES	YES	NO	NO	
CIARAOC	0200	CIARAOC	2015-06-01	JAMIE	NO	YES	YES	YES	NO	
DB@ADM	0200	DB@ADM	2022-08-30	SYS1	NO	YES	YES	NO	NO	
DB2ADM	0200	DB2ADM	2022-08-30	SYS1	NO	YES	YES	NO	NO	
DYMOKEL	0200	DYMOKEL	2010-03-10	LENNIE	NO	YES	YES	NO	NO	
GEOFF	0200	GEOFF	2011-11-11	LENNIE	NO	YES	NO	NO	NO	

```
[16]: r.users.find(SPECIAL='YES').skip(REVOKE='NO')
```

[16]:

_NAME	USBD_RECORD_TYPE	USBD_NAME	USBD_CREATE_DATE	USBD_OWNER_ID	USBD_ADSP	USBD_SPECIAL	USBD_OPER	USBD_REVOKE	USBD_GRPACC	USBD
CIARAOC	0200	CIARAOC	2015-06-01	JAMIE	NO	YES	YES	YES	NO	
IBMUSER	0200	IBMUSER	1995-06-06	IBMUSER	NO	YES	YES	YES	NO	

2 rows × 52 columns

Dataset Frames (1)

- r.dataset('SYS1.**')
  - 1 profile, if there is a match
- r.datasets
  - All dataset profiles
- r.datasets.find('SYS%.**')
  - Generic match with profile key

```
[19]: r.datasets.find('SYS%.**')
```

[19]:

_NAME	DSBD_RECORD_TYPE	DSBD_NAME	DSBD_VOL	DSBD_GENERIC	DSBD_CREATE_DATE	DSBD_C
SYS1.BRODCAST	0400	SYS1.BRODCAST		YES	2012-01-09	
SYS1.DFQP*	0400	SYS1.DFQP*		YES	2022-01-06	
SYS1.DFQ*	0400	SYS1.DFQ*		YES	2022-01-06	
SYS1.DGTCLIB	0400	SYS1.DGTCLIB		YES	2020-02-17	
SYS1.DGTMLIB	0400	SYS1.DGTMLIB		YES	2020-02-17	
SYS1.DGTPLIB	0400	SYS1.DGTPLIB		YES	2020-02-17	
SYS1.DGTSLIB	0400	SYS1.DGTSLIB		YES	2020-02-17	
SYS1.DGTTLIB	0400	SYS1.DGTTLIB		YES	2020-02-17	
SYS1.HRF*	0400	SYS1.HRF*		YES	2022-01-06	
SYS1.IMAGELIB	0400	SYS1.IMAGELIB		YES	2022-01-25	
SYS1.RACFDS	0400	SYS1.RACFDS		YES	2016-07-23	
SYS1.RACFDS.BACKUP	0400	SYS1.RACFDS.BACKUP		YES	2016-07-23	
[illegible]	0400	[illegible]		YES	2023-07-05	
SYS1.**.PAGE	0400	SYS1.**.PAGE		YES	2013-07-25	
SYS1.**	0400	SYS1.**		YES	2011-08-01	
SYS2.RACFDS	0400	SYS2.RACFDS		YES	2023-07-07	
SYS2.RACFDS.BACKUP	0400	SYS2.RACFDS.BACKUP		YES	2023-07-07	
SYS2.**	0400	SYS2.**		YES	2024-01-21	

30 rows × 35 columns

- r.datasets.find('SYS%.**', UACC='UPDATE')
  r.datasets.find('SYS%.**', UACC=['UPDATE','CONTROL','ALTER'])
  r.datasets.find('SYS%.**', UACC='UPDATE CONTROL ALTER'.split())
  r.datasets.find('SYS%.**').skip(UACC=['NONE','READ'])

Dataset Frames (2)

- Access for all user IDs
  - DSBD_UACC
    - Standard field in DSBD Frame
  - IDSTAR_ACCESS
    - ID(*) permits from DSACC Frame
  - ALL_USER_ACCESS
    - Highest value of the two
- r.datasets.match('SYS1.PARMLIB')
  - Best mathing profile

```
[30]: r.datasets.find('SYS1.**').skip(ALL_USER_ACCESS='NONE')[['DSBD_UACC','IDSTAR_ACCESS','ALL_USER_ACCESS']]
```

[30]:

_NAME	DSBD_UACC	IDSTAR_ACCESS	ALL_USER_ACCESS
SYS1.BRODCAST	NONE	READ	READ
SYS1.DFQP*	NONE	READ	READ
SYS1.DFQ*	NONE	READ	READ
SYS1.DGTCLIB	NONE	READ	READ
SYS1.DGTMLIB	NONE	READ	READ
SYS1.DGTPLIB	NONE	READ	READ
SYS1.DGTSLIB	NONE	READ	READ
SYS1.DGTTLIB	NONE	READ	READ
SYS1.HRF*	NONE	READ	READ
SYS1.SBLSCLI0	NONE	READ	READ
SYS1.SBLSKEL0	NONE	READ	READ
SYS1.SBLSMSG0	NONE	READ	READ
SYS1.SBLSPNL0	NONE	READ	READ
SYS1.SBLSTBL0	NONE	READ	READ
SYS1.SBPXEXEC	NONE	READ	READ
SYS1.SBPX*	NONE	READ	READ
SYS1.SCBD*	NONE	READ	READ
SYS1.SEDG*	NONE	READ	READ
SYS1.SERB*	NONE	READ	READ
SYS1.SMP.OTABLES	NONE	READ	READ
SYS1.V%%%%%%.**	NONE	READ	READ

Dataset Frames (3)

- r.datasets.match('SYS1.PARMLIB').acl()
  - Show permits from datasetAccess and datasetsConditionalAccess
- ...acl(resolve=True)
  - The user IDs that have access
- r.datasets.find('SYS1.**').acl(allows='UPDATE')
  - Who has more than READ
- r.datasets.find('SYS1.**').acl(resolve=True).find(USER_ID='ROB')
  - Scope of a user ID

```
[20]: r.datasets.match('SYS1.PARMLIB').acl()
```

[20]:

	NAME	VOL	USER_ID	AUTH_ID	ACCESS
0	SYS1.**		-group-	SYS1	ALTER
1	SYS1.**		LENNIE	LENNIE	ALTER
2	SYS1.**		TCPIP	TCPIP	READ

```
[23]: r.datasets.match('SYS1.PARMLIB').acl(resolve=True)
```

[23]:

	NAME	VOL	USER_ID	AUTH_ID	ACCESS
0	SYS1.**		9735	SYS1	ALTER
1	SYS1.**		ADCDMST	SYS1	ALTER
2	SYS1.**		ASH	SYS1	ALTER
3	SYS1.**		BILLA	SYS1	ALTER
4	SYS1.**		BPXOINIT	SYS1	ALTER

General Resource Frames

- General Resource profiles
  - Class + Resource name
  - Use both in find( ), skip( ) and match( )

```
[3]: r.generals.loc[['UNIXPRIV']]
```

[3]:

_CLASS_NAME	_NAME	GRBD_RECORD_TYPE	GRBD_NAME	GRBD_CLASS_NAME	GRBD_GENERIC
UNIXPRIV	SHARED.IDS	0500	SHARED.IDS	UNIXPRIV	NO
	SUPERUSER.FILESYS	0500	SUPERUSER.FILESYS	UNIXPRIV	NO
	SUPERUSER.FILESYS.CHOWN	0500	SUPERUSER.FILESYS.CHOWN	UNIXPRIV	NO
	SUPERUSER.FILESYS.PFSCTL	0500	SUPERUSER.FILESYS.PFSCTL	UNIXPRIV	NO

4 rows × 43 columns

- r.generals.find('UNIXPRIV')
- r.general.find('FACILITY').match('BPX.SUPERUSER')
  r.general.match('FACILITY','BPX.SUPERUSER')
- r.generals.find('TCICS*','CEMT')

Connect groups (1)

- 3 Frames with connect info:
  - GPMEM: connects
    - group, user ID, connect authority
  - USGCON: groupConnect
    - user ID, group
  - USCON: connectData
    - user ID, connect group name, group special, operations, auditor, last init date, revoke date, resume date
    - But not the connect authority (USE, CREATE, CONNECT, JOIN)
    - So pyRACF copies GPMEM_AUTH

Type	Prefix	DataFrame property	Description
0100	GPBD	groups	Group Basic Data
0101	GPSGRP	subgroups	Group Subgroups
~~0102~~	~~GPMEM~~	~~connects~~	~~Group Members~~
0103	GPINSTD	groupUSRDATA	Group Installation Data
0110	GPDFP	groupDFP	Group DFP Data
0120	GPOMVS	groupOMVS	Group OMVS Data
0130	GPOVM	groupOVM	Group OVM Data
0141	GPTME	groupTME	Group TME Data
0151	GPCSD	groupCSDATA	Group CSDATA custom fields
0200	USBD	users	User Basic Data
0201	USCAT	userCategories	User Categories
0202	USCLA	userClasses	User Classes
~~0203~~	~~USGCON~~	~~groupConnect~~	~~User Group Connections~~
0204	USINSTD	userUSRDATA	User Installation Data
0205	USCON	connectData	User Connect Data
0206	USRSF	userRRSFDATA	RRSF data
0207	USCERT	userCERTname	user certificate name
0208	USNMAP	userAssociationMapping	User Associated Mappings
0209	USDMAP	userDistributedIdMapping	User Associated Distributed Mappings
020A	USMFA	userMFAfactor	user Multifactor authentication data

Connect groups (2)

- r.connectData.find(USCON_GRP_SPECIAL='YES')
  - All group specials
- r.connectData.skip(GPMEM_AUTH='USE')
- r.connect('SYS1')
  - All user IDs connected to SYS1
- r.connect('**','IBMUSER')
  r.connect(userid='IBMUSER')
  - Groups that IBMUSER is member of

```
[38]: r.connect('SYS1')
```

[38]:

_NAME	USCON_RECORD_TYPE	USCON_NAME	USCON_GRP_ID	USCON_CONNECT_DATE	USCON_OWNER_ID
ADCDMST	0205	ADCDMST	SYS1	2004-11-10	SYS1
ASH	0205	ASH	SYS1	2011-05-31	LENNIE
BILLA	0205	BILLA	SYS1	2016-08-02	LENNIE
BPXOINIT	0205	BPXOINIT	SYS1	2002-11-13	IBMUSER
CEA	0205	CEA	SYS1	2009-11-17	IBMUSER
...	...	...	...	...	...
XHAB	0205	XHAB	SYS1	2019-09-24	LENNIE
ZFSAS	0205	ZFSAS	SYS1	2022-02-07	LENNIE

```
[37]: r.connect(None,'IBMUSER')
```

[37]:

_GRP_ID	USCON_RECORD_TYPE	USCON_NAME	USCON_GRP_ID	USCON_CONNECT_DATE	USCON_OWNER_ID
IZUADMIN	0205	IBMUSER	IZUADMIN	2019-11-11	ADCDMST
IZUUSER	0205	IBMUSER	IZUUSER	2019-11-11	ADCDMST
SYSCTLG	0205	IBMUSER	SYSCTLG	1995-06-06	IBMUSER
SYS1	0205	IBMUSER	SYS1	1995-06-06	IBMUSER
VSAMDSET	0205	IBMUSER	VSAMDSET	1995-06-06	IBMUSER

System related segments

- Uses the segment name as Frame name
  - from RDEFINE command
  - r.STDATA.stripPrefix()\
    .skip(TRUSTED='NO',PRIVILEGED='NO')
    - StripPrefix() removes STDATA_ from column names
  - r.CDTINFO
  - r.CFDEF
  - Etc
- Of course, match() works too

```
[41]: r.STDATA.stripPrefix().skip(TRUSTED='NO',PRIVILEGED='NO')
```

[41]:

_CLASS_NAME	_NAME	RECORD_TYPE	NAME	CLASS_NAME	USER_ID	GROUP_ID	TRUSTED	PRIVILEGED	TRACE
STARTED	CATALOG.*	0540	CATALOG.*	STARTED	START1	SYS1	YES	NO	NO
	DUMPSRV.*	0540	DUMPSRV.*	STARTED	START1	SYS1	YES	NO	NO
	GSKSRVR.**	0540	GSKSRVR.**	STARTED	GSKSRVR	SYS1	YES	NO	NO
	IEEVMPCR.*	0540	IEEVMPCR.*	STARTED	START1	SYS1	YES	NO	NO
	IXGLOGR.*	0540	IXGLOGR.*	STARTED	START1	SYS1	YES	NO	NO
	JES2.*	0540	JES2.*	STARTED	START1	SYS1	YES	NO	NO
	JES2S%%%.*	0540	JES2S%%%.*	STARTED	NETSRV	NJE	YES	NO	YES
	LLA.*	0540	LLA.*	STARTED	START1	SYS1	YES	NO	NO
	OMVS.*	0540	OMVS.*	STARTED	OMVSKERN	OMVSGRP	YES	NO	NO
	PAGENT.*	0540	PAGENT.*	STARTED	START1	SYS1	YES	NO	NO
	RACF.*	0540	RACF.*	STARTED	START1	SYS1	YES	NO	NO
	RACF.**	0540	RACF.**	STARTED	RACFSUB		YES	NO	YES
	SMF.*	0540	SMF.*	STARTED	START1	SYS1	YES	NO	NO
	TCPIP.*	0540	TCPIP.*	STARTED	TCPIP		YES	NO	NO
	TSO.*	0540	TSO.*	STARTED	START1	SYS1	YES	NO	NO
	VLF.*	0540	VLF.*	STARTED	START1	SYS1	YES	NO	NO
	VTAM.*	0540	VTAM.*	STARTED	START1	SYS1	YES	NO	NO
	XCFAS.*	0540	XCFAS.*	STARTED	START1	SYS1	YES	NO	NO

Rule-based Tests

Test for compliance?

- Answer these questions quickly:
  - Are there any profiles with UACC>NONE
    - Some exceptions…
  - Have the administrators given ID(*) ACCESS(UPDATE)
  - Are all IDs in profile fields still defined
    - Orphan permits
    - Orphan OWNER values
  - RACF administrator is OWNER of profile, or ALTER permit?
- Fix and keep testing

With pyRACF Frames

```
def orphans(self):

    if self._records[self.DSACC_RECORDTYPE]['parsed'] + self._records[self.GRACC_RECORDTYPE]['parsed'] == 0:
        raise StoopidException('No dataset/generic access records parsed! (PEBKAM/ID-10T error)')

    datasetOrphans = None
    genericOrphans = None

    if self._records[self.DSACC_RECORDTYPE]['parsed'] > 0:
        self._datasetAccess = self._datasetAccess.assign(inGroups=self._datasetAccess.DSACC_AUTH_ID.isin(self._groups.GPBD_NAME))
        self._datasetAccess = self._datasetAccess.assign(inUsers=self._datasetAccess.DSACC_AUTH_ID.isin(self._users.USBD_NAME))
        datasetOrphans = self._datasetAccess.loc[(self._datasetAccess['inGroups'] == False) & (self._datasetAccess['inUsers'] == False)
                                                 & (self._datasetAccess['DSACC_AUTH_ID'] != '*') & (self._datasetAccess['DSACC_AUTH_ID'] != '&RACUID')]

    if self._records[self.GRACC_RECORDTYPE]['parsed'] > 0:
            self._genericAccess = self._genericAccess.assign(inGroups=self._genericAccess.GRACC_AUTH_ID.isin(self._groups.GPBD_NAME))
            self._genericAccess = self._genericAccess.assign(inUsers=self._genericAccess.GRACC_AUTH_ID.isin(self._users.USBD_NAME))
            genericOrphans =  self._genericAccess.loc[(self._genericAccess['inGroups'] == False) & (self._genericAccess['inUsers'] == False)
                                                      & (self._genericAccess['GRACC_AUTH_ID'] != '*') & (self._genericAccess['GRACC_AUTH_ID'] != '&RACUID')]

    return datasetOrphans, genericOrphans
```

With pyRACF rules Class

```
v_permit = r.rules
v_permit.load(rules='''

dataset orphans:
  - [DSACC,DSCACC]
  - test:
      field: AUTH_ID
      fit: ACLID

''').verify()
```

- v_permit is rules (verifier) instance
- rules is YAML string with
  - Description of rule
  - Tables to check (prefix name)
  - Test criteria
    - Field name
      - Don’t bother about prefix
    - Field value
      - Literal, list of literals, or
      - Member of a domain: ACLID

Predefined domains

- Built-in
  - 'USER': r.users.index
  - 'GROUP': r.groups.index,
  - 'SPECIAL': ['*','&RACUID','&RACGRP']
  - 'DELETE': ['']
  - 'ID': r.users.index.union(r.groups.index)
  - 'ACLID':_domains['SPECIAL'].union(_domains['ID'])
  - 'RACFVARS': r.generals.find('RACFVARS').index.get_level_values(1)
  - ‘USERQUAL': r.users.index.union(_domains['RACFVARS'])
  - 'CATEGORY': r.generalMembers.find('SECDATA','CATEGORY')['GRMEM_MEMBER'].values
  - 'SECLEVEL': r.generalMembers.find('SECDATA','SECLEVEL')['GRMEM_MEMBER'].values
  - SECLABEL': r.generals.find('SECLABEL').index.get_level_values(1)
- Add your own
  - rv_instance.add_domains({‘Our_DBAs’: r.connect(‘SYSDBA’).index})

Starter set rules

```
orphan permits:
 - [DSACC, DSCACC, DSMEM, GRACC, GRCACC]
 - test:
     field: AUTH_ID
     fit: ACLID

ID references in resouce profiles:
 - [DSBD, GRBD]
 - test:
   - field: NOTIFY_ID
     fit: USER
     rule: notify on dataset and resource profiles must be user
   - field: OWNER_ID
     fit: ID
     rule: owner must be user or group

general resource profile key qualifiers:
 - GRBD
 - class: JESSPOOL
   match: '\S+?.(id).\S+'
   test:
     field: id
     fit: USERQUAL
     value: ['*','+MASTER+']
   rule: 2nd qualifier in JESSPOOL should be a user ID
 - class: SURROGAT
   match: (id).SUBMIT
   test:
     field: id
     fit: USERQUAL
```

- Dictionary (dict) with rules
  - Rule description as key
  - Table name (or list)
  - Criteria (one or more):
    - Select or exclude
    - Match fields from profile key
    - One or more tests
      - Field name
      - Field value, or
      - Field fit to domain
  - repeat

YAML ……..

- YAML is not a markup language
  - Builds lists and dicts from text files
  - Don’t bother about quotes, parens, brackets, braces, commas
  - Aggressively interprets text as flag or number
    - countries: dk, se, no, fi
      → [‘dk’,’se’,False,’fi’]
    - USBD_SPECIAL: YES
      → ‘USBD_SPECIAL’ : True
    - Subnetwork: 192.168
      → float(192.168)
- When in doubt, quote your strings
- https://ruudvanasseldonk.com/2023/01/11/the-yaml-document-from-hell