



Introduction to Writing Micro-services

2010 iRODS User Group Meeting

mwan@dicerresearch.org

Micro-service Input/output parameters



- Prototype of a micro-service

```
int findObjType ( msiParam_t *objInput ,  
                 msiParam_t *typeOutput ,  
                 ruleExecInfo_t *rei );
```

- All micro-services only use msiParam_t for input/output
- The last input parameter is always ruleExecInfo_t *rei

Micro-service input/output parameter type - msParam_t



All MS input/output parameters use the same structure

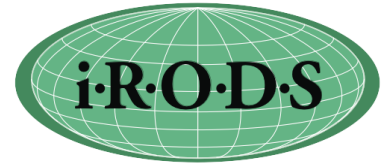
```
typedef struct MsParam {  
    char *label;          /* name of the parameter */  
    char *type;          /* type of the parameter */  
    void *inOutStruct;   /* pointer to value/structure of  
                          the parameter */  
    bytesBuf_t *inpOutBuf; /* optional buffer pointer  
                          for binary values */  
} msParam_t;
```

- label
 - Used by the rule engine to identify the parameter
 - Not a concern for MS programming
- type
 - Identifies the type of data stored in inOutStruct
 - Self defining data structure
- inOutStruct
 - pointer to a struct that contains the input/output data
- inpOutBuf
 - Pointer to an optional buffer for large data



The type field of msParam_t

- The “type” field defines the type of data in the struct.
- Some commonly used types:
 - STR_MS_T - string type (most common)
 - StrArray_MS_T
 - INT_MS_T – integer type
 - IntArray_MS_T
 - DOUBLE_MS_T
 - DataObjInp_MS_T– input struct for data object operation
 - CollInp_MS_T
 - KeyValPair_MS_T – key/value pair
 - GenQueryInp_MS_T – input struct for general query
 - GenQueryOut_MS_T
 - RodsObjStat_MS_T
- Defined in msParam.h



msParam helper routines

- Routines to parse and fill in the msParam_t struct
 - Can be found in msParam.c
 - Int fillMsParam (msParam_t *msParam, char *label, char *type, void *inOutStruct, bytesBuf_t *inpOutBuf);
 - Generally, fields will only be modified if non-null input. Normally, "label" input is null.
 - Int fillIntInMsParam (msParam_t *msParam, int myInt);
 - Int fillStrInMsParam (msParam_t *msParam, char *myStr);
 - Int resetMsParam (msParam_t *msParam);
 - Free all fields except label.
 - Int parseMspForPosInt (msParam_t *inpParam);
 - char *parseMspForStr (msParam_t *inpParam);
 - Int parseMspForCollInp (msParam_t *inpParam, collInp_t *collInpCache, collInp_t **outCollInp, int writeToCache)

msKeyValStr – string input for key/value



- A special kind of STR_MS_T
- Format – keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
- A way to input several inputs with one string
- Helper routines – parseMsKeyValStrForDataObjInp and parseMsKeyValStrForCollInp
- Example:
 - `validKwFlags = DEST_RESC_NAME_FLAG | CREATE_MODE_FLAG | DATA_TYPE_FLAG | FORCE_FLAG_FLAG | DATA_SIZE_FLAGS | OBJ_PATH_FLAG;`
 - `rei->status = parseMsKeyValStrForDataObjInp (msKeyValStr, myDataObjInp, DEST_RESC_NAME_KW, validKwFlags, &outBadKeyWd);`
 - 3rd input (DEST_RESC_NAME_KW) – for backward compatibility. If “=” not present in msKeyValStr, assume it is a destRescName input

Session system parameters



- ruleExecInfo_t *rei
 - A large data structure passed when invoking a rule
 - Contains system parameters and parameters relevant to the rule invoked:
 - *rsComm - client-server communication structure
 - *doi - dataObject information
 - *rescGrp - resource (group) informations
 - *uoinc - client user information

Session system parameters



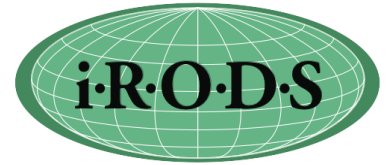
- \$ variables – Variables start with “\$”
 - Provides a way for rules to reference values in rei structure
 - A mapping from a name to values in rei.
 - These mappings are defined in a configuration file:
 - objPath rei->doi->objPath
 - rescName rei->doi->rescName
 - userNameClient rei->uoic->userName
 - These variables can be referenced by rules and MSs
 - Condition:
 \$objPath like /zone/home/sekar@sdsc/nvo/*
 - Input Parameters of MS: findObjType(\$objName,*Type)

Writing Micro-services



- Typically MS codes are short
- Call existing server routines
 - Reasonably familiar with server routines
 - Server API handler routines
 - Each client API has one server API handler
 - |
n
dataObjOpen.h : rcDataObjOpen() and rsDataObjOpen()
 - To open an iRods file on the server, call rsDataObjOpen
 - Prototype of Client API
s and Server API handler are given in the lib/api/include directory

A micro-service example (msiDataObjRepl in reDataObjOpr.c



```
iint
msiDataObjRepl
  (msParam_t
   *i
   inpParam1, msParam_t *msKeyValStr, msParam_t *outParam, ruleExecInfo_t *rei) {
  rsComm_t *rsComm;
  dataObjInp_t dataObjInp, *myDataObjInp;
  transStat_t *transStat = NULL;
  char *outBadKeyWd;
  int validKwFlags;
  RE_TEST_MACRO ("  Calling msiDataObjRepl")
  rsComm = rei->rsComm;
  /* parse inpParam1 */

  re
  i->status = parseMspForDataObjInp (inpParam1, &dataObjInp, &myDataObjInp, 0);
  if (rei->status < 0) {
    ....
  }
```



Micro-service example (cont.)

```
validKwFlags = OBJ_PATH_FLAG | DEST_RESC_NAME_FLAG | NUM_THREADS_FLAG  
|  
  BACKUP_RESC_NAME_FLAG | RESC_NAME_FLAG | UPDATE_REPL_FLAG |  
  REPL_NUM_FLAG | ALL_FLAG | IRODS_ADMIN_FLAG | VERIFY_CHKSUM_FLAG |  
  RBUDP_TRANSFER_FLAG | RBUDP_SEND_RATE_FLAG |  
RBUDP_PACK_SIZE_FLAG;
```

```
rei->status = parseMsKeyValStrForDataObjInp (msKeyValStr, myDataObjInp,  
  DEST_RESC_NAME_KW, validKwFlags, &outBadKeyWd);
```

```
if (rei->status < 0) { ...}  
rei->status = rsDataObjRepl (rsComm, myDataObjInp, &transStat);  
if (rei->status >= 0) {  
  fillIntInMsParam (outParam, rei->status);  
} else {.....}  
return (rei->status);  
}
```



Writing micro-services

Adding a MS to the built-in server module

- Add a MS routine `msiMyRoutine` to an existing file `reDataObjOpr.c`
- Add the prototype of `msiMyRoutine` to `reDataObjOpr.h`
 - `int msiMyRoutine (msParam_t *collection, msParam_t *targetResc, msParam_t *status, ruleExecInfo_t *rei);`
 - Add a line to the `reAction.table` file

.....

```
{"msiRmColl",3,(funcPtr) msiRmColl},  
{"msiReplColl",4,(funcPtr) msiReplColl},  
{"msiMyRoutine",3,(funcPtr) msiMyRoutine},
```

Adding a new Micro-service module



- Modules are a set of optional MSs that can be compiled and linked with the server
- https://www.irods.org/index.php/How_to_create_a_new_module
- The “modules” directory contains the optional MS modules
 - hdf5, images, ERA
- Create a new directory for your module
 - Easiest just to copy the entire directory of an existing module for the structure
- Edit the Makefile to include your MS files
- Build the server with your module, do either:
 - `./configure --enable-myModule`
 - Edit the `config/config.mk` file by add an entry in the MODULES definition
 - `MODULES= properties hdf5 myModule`