Resources

Resources are the actual external tasks that are controlled by the Resource Manager. The resources provide SIGN ON, SIGN OFF and possibly other messages. The resource instances can be sent CONNECT, DISCONNECT, CONTROL and possibly other messages. They can provide DISCONNECT and possibly other messages themselves. A message interface is used.

JEXEC DEFINITIONS
The Resource Manager is specified in rmtaskid.h as JEXEC_TASKID_RESOURCE_MANAGER.

MESSAGE DEFINITIONS
SIGN ON (Resource to Resource Manager) - Mandatory
Indicates to the Resource Manager that a Resource is available for operation. This identifies the Resource, provides Version and Naming information [checking purposes] etc.

Field	**Value**	**Notes**
Type	M_STATUS_IND	
SubType	SUBTYPE_RM_RESOURCE_SIGNON	
Param(0)	(Any)	
Contents	PID_RM_RESOURCE	(Mandatory)
	PID_RM_RESOURCE_NAME	(Optional)
	PID_RM_RESOURCE_VERSION	(Optional)
	PID_RM_RESOURCE_COPYMASK	(Optional)
	PID_RM_RESOURCE_MODULE	(Obsolete)
	(Any; SignOn Information)	(Optional)

SIGN OFF (Resource to Resource Manager) - Optional
Indicates to the Resource Manager that a Resource is no longer available for operation. This identifies the particular Resource, and may provide additional diagnostic information.

Field	**Value**	**Notes**
Type	M_STATUS_IND	
SubType	SUBTYPE_RM_RESOURCE_SIGNOFF	
Param(0)	(Any)	
Contents	PID_RM_RESOURCE	(Mandatory)
	(Any, Diagnostic Information)	(Optional)

CONNECT REQUEST (Resource Manager to Resource) - Mandatory
Indicates to the Resource that a particular Instance should commence operation. This may contain Configuration and Initialisation Information.

Field	**Value**	**Notes**
Type	M_CONNECT_REQ	
SubType	SUBTYPE_RM_RESOURCE_CONTROL	
Param(0)	Resource Copy (> 0)	(Specific Instance)
Contents	PID_RM_RESOURCE_PEER_LOWER	(Optional)
	PID_RM_RESOURCE_PEER_UPPER	(Optional)
	PID_RM_RESOURCE_PEER_MGMT	(Optional)
	(Any, Config/Init Information)	

DISCONNECT REQUEST (Resource Manager to Resource) - Mandatory
Indicates to the Resource that a particular Instance should stop operation.

Field	**Value**	**Notes**
Type	M_DISCONNECT_REQ	
SubType	SUBTYPE_RM_RESOURCE_CONTROL	
Param(0)	Resource Copy (> 0)	(Specific Instance)

Contents	(Any, Disconnect Information)	

CONTROL REQUEST (Resource Manager to Resource) - Mandatory
Provides the Resource with control type information that has originated from the Resource Manager (a subtype other than SUBTYPE_RM_RESOURCE_INFO will have originated from an entity other than the Resource Manager).

Field	**Value**	**Notes**
Type	M_CONTROL_REQ	
SubType	SUBTYPE_RM_RESOURCE_CONTROL, or (Any)	(Info)
Param(0)	Resource Copy (> 0), or Resource Copy (= 0)	(Specific Instance) (General Resource)
Contents	PID_RM_RESOURCE_PEER_LOWER	(Optional)
	PID_RM_RESOURCE_PEER_UPPER	(Optional)
	PID_RM_RESOURCE_PEER_MGMT	(Optional)
	(Any, Config/Init/Control Information)	

DISCONNECT INDICATION (Resource to Resource Manager) - Mandatory
Provides the Resource Manager with an indication that the particular Instance has failed operating. This may contain Disconnect Information.

Field	**Value**	**Notes**
Type	M_DISCONNECT_IND	
SubType	SUBTYPE_RM_RESOURCE_CONTROL	(Optional)
Param(0)	Resource Copy (> 0)	(Specific Instance)
Contents	(Any, Disconnect Information)	

PID DEFINITIONS
The following PIDs are defined for this interface.

Literal	**Value**	**Contents**
PID_RM_RESOURCE	PID_GLOBAL_BASE + 0x07	tResourceInfo
PID_RM_RESOURCE_NAME	PID_TASK_BASE + 0x01	tResourceName
PID_RM_RESOURCE_VERSION	PID_TASK_BASE + 0x02	tResourceVersion
PID_RM_RESOURCE_COPYMASK	PID_TASK_BASE + 0x03	tResourceMask (or part, thereof)
PID_RM_RESOURCE_NOTIFY_TAG	PID_TASK_BASE + 0x04	u_byte_t
PID_RM_RESOURCE_AVAILABLE_COUNT	PID_TASK_BASE + 0x05	u_byte_t
PID_RM_RESOURCE_PEER_LOWER	PID_TASK_BASE + 0x10	tResourceInfo
PID_RM_RESOURCE_PEER_UPPER	PID_TASK_BASE + 0x11	tResourceInfo
PID_RM_RESOURCE_PEER_MGMT	PID_TASK_BASE + 0x12	tResourceInfo
PID_RM_RESOURCE_ACTIVATE	PID_TASK_BASE + 0x20	boolean_t
PID_RM_RESOURCE_DEACTIVATE	PID_TASK_BASE + 0x21	boolean_t
PID_RM_RESOURCE_TIMESLOT_ID	PID_TASK_BASE + 0x30	tResourceInterfaceId
PID_RM_RESOURCE_TIMESLOT_CHANNEL	PID_TASK_BASE + 0x31	tTimeslotChannel []

PID_RM_RESOURCE_TIMESLOT_MASK	PID_TASK_BASE + 0x32	tTimeslotMask []
PID_RM_RESOURCE_TIMESLOT_INFO	PID_TASK_BASE + 0x33	u_byte_t []
PID_RM_RESOURCE_SOFTWARE_ID	PID_TASK_BASE + 0x38	tResourceInterfaceId
PID_RM_RESOURCE_SOFTWARE_INFO	PID_TASK_BASE + 0x39	u_byte_t []

DATA DEFINITIONS

The following data types are defined for this interface.

Name	Value	Description
tResourceId	u_short_t	Identifier for the Resource, corresponds to JEXEC Task Id
tResourceCopy	u_byte_t	Copy for the Resource, 0 addresses the Resource, and 1 to 255 (inclusive) addresses an Instance.
tResourceVersion	u_short_t	Version for the Resource 0xMMmm, (e.g. V1.01 = 0x0101
tResourceName	u_byte_t [32]	Name for the Resource (e.g. "QUICC QMC Driver")

tResourceInfo	struct {
	tResourceId Id;
	tResourceCopy Copy;
	tModuleId Id;
	};
		Structure to bind together a Id and Copy for transport in PID.

tResourceMask	u_byte_t [1 ... 64];	Set of bits to indicate which Copies are available or not.
tResourceInterfaceId	u_byte_t	A byte to identify a unique interface at a resource.

API DEFINITIONS

The following library API definitions are defined for this interface.

```
#   ifndef      _RM_LIB_H_
#   define      _RM_LIB_H_

#   ifdef       __cplusplus
    extern "C"
      {
#   endif

/*  ------------------------------------------------------------------ */

#   include     <jexec.h>
#   include     <jtecstd.h>
#   include     <rm_res.h>
#   include     <rm_ts.h>

/*  ------------------------------------------------------------------ */

typedef u_byte_t tResourceInterfaceId;

/*  ------------------------------------------------------------------ */

/* Encode and Decode Resource Info
 * into a PID_RM_RESOURCE
 *  msg -- msg to encode or decode
 *  info -- pointer to the resource information
 */
boolean_t rm_EncodeResourcePID (MSGPTR msg, const tResourceInfo* info);
boolean_t rm_DecodeResourcePID (const MSGPTR msg, tResourceInfo* info);


/* Encode and Decode Resource Name
 * into a PID_RM_RESOURCE_NAME
 *  msg -- msg to encode or decode
 *  nameSz -- size of the name
```

```
 *  name -- the name
 */
boolean_t rm_EncodeResourceNamePID (MSGPTR msg, const char* name);
boolean_t rm_DecodeResourceNamePID (const MSGPTR msg, int* nameSz, char* name);


/* Encode and Decode Resource Vers
 * into a PID_RM_RESOURCE_VERSION
 *  msg -- msg to encode or decode
 *  version -- the resource version
 */
boolean_t rm_EncodeResourceVersPID (MSGPTR msg, const tResourceVersion version);
boolean_t rm_DecodeResourceVersPID (const MSGPTR msg, tResourceVersion* version);


/* Encode and Decode Resource Mask
 * into a PID_RM_RESOURCE_MASK
 *  msg -- msg to encode or decode
 *  maskSz -- size of the mask
 *  mask -- the mask
 */
boolean_t rm_EncodeResourceMaskPID (MSGPTR msg, const int maskSz, const tResourceMask*
mask);
boolean_t rm_DecodeResourceMaskPID (const MSGPTR msg, int* maskSz, tResourceMask*
mask);


/* Encode and Decode Resource Lower Layer
 * into a PID_RM_RESOURCE_PEER_LOWER
 *  msg -- msg to encode or decode
 *  info -- the lower layer resource
 */
boolean_t rm_EncodeLowerLayer (MSGPTR msg, const tResourceInfo* info);
boolean_t rm_DecodeLowerLayer (const MSGPTR msg, tResourceInfo* info);


/* Encode and Decode Resource Upper Layer
 * into a PID_RM_RESOURCE_PEER_UPPER
 *  msg -- msg to encode or decode
 *  info -- the upper layer resource
 */
boolean_t rm_EncodeUpperLayer (MSGPTR msg, const tResourceInfo* info);
boolean_t rm_DecodeUpperLayer (const MSGPTR msg, tResourceInfo* info);


/* Encode and Decode Resource Mgmt Layer
 * into a PID_RM_RESOURCE_PEER_MGMT
 *  msg -- msg to encode or decode
 *  info -- the mgmt layer resource
 */
boolean_t rm_EncodeMgmntLayer (MSGPTR msg, const tResourceInfo* info);
boolean_t rm_DecodeMgmntLayer (const MSGPTR msg, tResourceInfo* info);


/* Encode and Decode Resource Interface
 * into Resource Software Interface PIDs
 *  msg -- msg to encode or decode
 *  activated -- if the interface is activated or deactivated
 *  node_local -- the local interface's resource
 *  id_local -- the local interface's id
 *  node_remote -- the remote interface's resource
 *  id_remote -- the remote interface's id
 */
boolean_t rm_EncodeResourceIfSoftware (MSGPTR msg, const boolean_t activated, const
tResourceInfo* node_local, const tResourceInterfaceId id_local, const tResourceInfo*
node_remote, const tResourceInterfaceId id_remote);
boolean_t rm_DecodeResourceIfSoftware (const MSGPTR msg, boolean_t* activated,
tResourceInfo* node_local, tResourceInterfaceId* id_local, tResourceInfo* node_remote,
tResourceInterfaceId* id_remote);


/* Encode and Decode Resource Interface
 * into Resource Timeslot Interface PIDs
 *  msg -- msg to encode or decode
 *  activated -- if the interface is activated or deactivated
 *  node_local -- the local interface's resource
 *  id_local -- the local interface's id
 *  chanSz_local -- the local interface's channel size
 *  chanLst_local -- the local interface's channel list
 *  maskLst_local -- the local interface's mask list
 *  node_remote -- the remote interface's resource
```

```
 *  id_remote -- the remote interface's id
 *  chanSz_remote -- the remote interface's channel size
 *  chanLst_remote -- the remote interface's channel list
 *  maskLst_remote -- the remote interface's mask list
 */
boolean_t rm_EncodeResourceIfTimeslot (MSGPTR msg, const boolean_t activated, const
tResourceInfo* node_local, const tResourceInterfaceId id_local, const int
chanSz_local, const tTimeslotChannel* chanLst_local, const tTimeslotMask*
maskLst_local, const tResourceInfo* node_remote, const tResourceInterfaceId id_remote,
const int chanSz_remote, const tTimeslotChannel* chanLst_remote, const tTimeslotMask*
maskLst_remote);
boolean_t rm_DecodeResourceIfTimeslot (const MSGPTR msg, boolean_t* activated,
tResourceInfo* node_local, tResourceInterfaceId* id_local, int* chanSz_local,
tTimeslotChannel* chanLst_local, tTimeslotMask* maskLst_local, tResourceInfo*
node_remote, tResourceInterfaceId* id_remote, int* chanSz_remote, tTimeslotChannel*
chanLst_remote, tTimeslotMask* maskLst_remote);


/* Sign On or Off to the Resource Manager
 * taskId -- the task hosting this resource
 * resourceId -- the resource identifier
 * resourceCopy -- the resource copy available
 * moduleId -- the module identifier
 * resourceName -- the resource name
 * resourceVers -- the resource version
 */
boolean_t rm_IssueSignOnMessage (const u_short_t taskId, const tResourceId resourceId,
const tResourceCopy resourceCopy, const tModuleId resourceModule, const char*
resourceName, const tResourceVersion resourceVers);
boolean_t rm_IssueSignOffMessage (const u_short_t taskId, const tResourceId
resourceId, const tResourceCopy resourceCopy, const tModuleId resourceModule, const
char* resourceName, const tResourceVersion resourceVers);

/*  ------------------------------------------------------------------ */

#   ifdef       __cplusplus
      }
#   endif

#   endif       /*_RM_LIB_H_*/
```

CLIENT DEFINITIONS

The following is an example of processing at a client.

```
...

#include    <jexec.h>
#include    <jtecstd.h>
#include    <j5000.h>

...

#   ifdef  RM_RESMGMT_ENABLED
#include    <rm_res.h>
#include    <rm_res_l.h>
#include    <rm_res_n.h>
#   endif

...

#   ifdef   RM_RESMGMT_ENABLED

tResourceInfo   SomeTask_Resource_LowerLayer [_TASK_COPY];
tResourceInfo   SomeTask_Resource_UpperLayer [_TASK_COPY];
boolean_t       SomeTask_Resource_Active [_TASK_COPY];

boolean SomeTask_Resource_SignOn (void)
  {
    tResourceCopy copy;

#   ifdef   RM_RESMGMT_DEBUG
    jprintf ("SomeTask (): Resource Sign On");
#   endif

    /*
     * Resource Module:
     *  Sign On to the Resource Manager to let it know that
     *  the SomeTask Task is present.
     */
```

```
    if (rm_IssueSignOnMessage (_TASK_ID,
                               RM_RESOURCE_ID_SW_SomeTask,
                               _TASK_COPY,
                               0,
                               _TASK_NAME,
                               _TASK_VERS) == false)
        return FALSE;
    for (copy = 0; copy < _TASK_COPY; copy++)
        SomeTask_Resource_Active [copy] = false;

    return TRUE;
  }

boolean SomeTask_Resource_SignOff (void)
  {
    tResourceCopy copy;

#   ifdef   RM_RESMGMT_DEBUG
    jprintf ("SomeTask (): Resource Sign Off");
#   endif

    /*
     * Resource Module:
     *  Sign Off to the Resource Manager to let it know that
     *  the SomeTask Task is not present.
     */
    if (rm_IssueSignOffMessage (_TASK_ID,
                               RM_RESOURCE_ID_SW_SomeTask,
                               _TASK_COPY,
                               0,
                               _TASK_NAME,
                               _TASK_VERS) == false)
        return FALSE;
    for (copy = 0; copy < _TASK_COPY; copy++)
        SomeTask_Resource_Active [copy] = false;

    return TRUE;
  }

boolean SomeTask_Resource_Configure (tResourceCopy copy, MSGPTR msg)
  {
    tResourceInfo info;

    if (copy < 1 || copy > _TASK_COPY ||
        SomeTask_Resource_Active [copy] == true)
        return FALSE;

#   ifdef   RM_RESMGMT_DEBUG
    jprintf ("SomeTask (): Resource Configure(%d)", copy);
#   endif

    /* Configure Components */
    if (rm_DecodeLowerLayer (msg, &SomeTask_Resource_LowerLayer [copy]) == false)
        return FALSE;
    if (rm_DecodeUpperLayer (msg, &SomeTask_Resource_UpperLayer [copy]) == false)
        return FALSE;

    return TRUE;
  }

boolean SomeTask_Resource_Connect (tResourceCopy copy)
  {
    if (copy < 1 || copy > _TASK_COPY ||
        SomeTask_Resource_Active [copy] == true)
        return FALSE;

#   ifdef   RM_RESMGMT_DEBUG
    jprintf ("SomeTask (): Resource Connect(%d)", copy);
#   endif

    /* Initialise Components */
    SomeTask_Resource_Active = true;

    return TRUE;
  }

boolean SomeTask_Resource_Disconnect (tResourceCopy copy)
  {
    if (copy < 1 || copy > _TASK_COPY ||
        SomeTask_Resource_Active [copy] == false)
        return FALSE;
```

```
#   ifdef   RM_RESMGMT_DEBUG
    jprintf ("SomeTask (): Resource Disconnect(%d)", copy);
#   endif

    /* Terminate Components */
    SomeTask_Resource_Active = false;

    return TRUE;
  }

boolean SomeTask_Resource_Fail (tResourceCopy copy)
  {
    MSGPTR msg;

    if (copy < 1 || copy > _TASK_COPY)
        return FALSE;

#   ifdef   RM_RESMGMT_DEBUG
    jprintf ("SomeTask (): Resource Fail(%d)", copy);
#   endif

    /* Terminate Components */
    SomeTask_Resource_Active = false;

    /* Fail */
    if ((msg = NewMessage (_TASK_ID)) == NULL)
        return FALSE;
    msg->type = M_DISCONNECT_IND;
    msg->parameter [0] = copy;
    if (SendMessage (JEXEC_TASKID_RESOURCE_MANAGER, msg) == FALSE)
      {
        ReturnMessage (_TASK_ID, msg);
        return FALSE;
      }

    return TRUE;
  }

#   endif   /*RM_RESMGMT_ENABLED*/

...

void SomeTask_main (void)
  {
    ...

#   ifdef  RM_RESMGMT_ENABLED
    while (SomeTask_Resource_SignOn () == FALSE)
        Relinquish ();
#   endif  /*RM_RESMGMT_ENABLED*/

    ...

    switch (msg->type)
      {
#   ifdef   RM_RESMGMT_ENABLED
        case M_CONNECT_REQ:
            if (SomeTask_Resource_Configure (msg->parameter [0], msg) == FALSE)
              {
                SomeTask_Resource_Fail (msg->parameter [0]);
                break;
              }
            if (SomeTask_Resource_Connect (msg->parameter [0]) == FALSE)
              {
                SomeTask_Resource_Fail (msg->parameter [0]);
                break;
              }
            break;
        case M_DISCONNECT_REQ:
            if (SomeTask_Resource_Disconnect (msg->parameter [0]) == FALSE)
                break;
            break;
#   endif  /*RM_RESMGMT_ENABLED*/

    ...

#   ifdef  RM_RESMGMT_ENABLED
    while (SomeTask_Resource_SignOff () == FALSE)
        Relinquish ();
#   endif  /*RM_RESMGMT_ENABLED*/
```

```
    . . .
}
```

NOTES

1. More messages than those listed above may be used (e.g. CONTROL INDICATION, CONTROL CONFIRMATION, STATUS INDICATION .. etc); however these are out of the scope of the basic Resource Manager.
2. The CONTROL REQUEST can be sent to a Resource for two reasons; either generated internally (e.g. Service Manager or Connection Manager information), or generated externally (e.g. operation from a Port Manager). Only the former will have the sub type.
3. The SUBTYPE_RM_RESOURCE_CONTROL SubType is intended to provide a definite indication of the message's purpose. It could be removed.