Broadcom Management API

Revision 7.19.0 • Date May 7 2010

Prepared by: Hao-Yang Feng

Broadcom Corporation
5300 California Avenue
Irvine CA 92617
www.broadcom.com

1. Revision History

Version	Date	Author	Comments
2.2.3	12/12/01	Hao-Yang Feng	1. Add new diagnostic APIs that will return error code instead of returning an ASCII string. 2. BmapiInitDiag() and BmapiUnInitDiag() are required to call before and after doing diagnostic function.
2.2.4	12/14/01	Hao-Yang Feng	Modify API description for BmapiInitDiag() and BmapiUnInitDiag().
2.2.6	1/16/02	Hao-Yang Feng	Add BmapiGetFirmwareInfo().
2.2.7	1/17/02	Hao-Yang Feng	Modify BM_ASF_CFG structure.
2.2.8	1/31/02	Hao-Yang Feng	1. Add BmapiWriteFirmware(). 2. Add more data in BM_ADAPTER_INFO_EX.
2.2.9	2/8/02	Hao-Yang Feng	Add BmapiReadFirmware().
2.2.10	2/15/02	Hao-Yang Feng	Add ASF configuration change event.
2.2.13	2/22/02	Hao-Yang Feng	1. Modify BmapiGetPnpDevId(). 2. Add BmapiGetSystemASFTables().
2.2.15	3/7/02	Hao-Yang Feng	1. Modify BmapiGetPhyNic(). 2. Modify BmapiGetASFTable(). 3. Modify BmapiSetASFTable(). 4. Modify BmapiGetFirmwareInfo(). 5. Modify BmapiGetSystemASFTables().
2.2.16	3/12/02	Hao-Yang Feng	Add BmapiGetBrcmVirNic().
2.2.19	4/3/02	Hao-Yang Feng	1. Add BmapiReadNicMem(). 2. Add BmapiWriteNicMem(). 3. Add BmapiTestASF().
2.2.22	6/18/02	Hao-Yang Feng	1. Add BmapiGetIpAddrInfo() 2. Add BmapiTestNetwork() 3. Add BmapiGetPowerMode() 4. Add BmapiSetPowerMode() 5. Modify BM_FW_NIC_HW_CONFIG
2.2.23	6/25/02	Hao-Yang Feng	1. Add BmapiGetBRCMNicInfoEx() 2. Add BmapiGetLastDiagPort()
3.0.0	8/8/02	Hao-Yang Feng	1. Add BmapiWriteFirmwareInfo() 2. Modify BM_ASF_TABLE 3. Modify BM_BRCM_ADAPTER_INFO_EX 4. Modify BM_FW_MEDIA_MANUFACT_REGION
3.0.2	8/23/02	Hao-Yang Feng	Modify BM_ASF_MISC.
3.0.3	8/27/02	Hao-Yang Feng	1. Modify section “How to do diagnoctics?”. 2. Modify BmapiGetASFTable(). 3. Modify BmapiSetASFTable().

			4. Modify BmapiGetFirmwareInfo(). 5. Modify BmapiGetPowerMode(). 6. Modify BmapiSetPowerMode(). 7. Modify BmapiWriteFirmwareInfo().
3.0.7	9/24/02	Hao-Yang Feng	Add 'MaxSpeed' in BM_BRCM_ADAPTER_INFO_EX.
3.0.9	10/16/02	Hao-Yang Feng	Add BmapiGetTTBRCMNicInfo()
3.0.10	10/21/02	Hao-Yang Feng	Add supported network adapter information to each API.
3.0.13	11/8/02	Hao-Yang Feng	1. Add BmapiSetPHYStatus(). 2. Add BmapiGetPHYStatus() 3. Add ‘DisablePowerSaving’ bit.
6.1.0	3/19/03	Hao-Yang Feng	Add BmapiRetrieveLinkStatusEx().
6.1.2	5/5/03	Hao-Yang Feng	1. Add BmapiTestLEDsEx2(). 2. Support PXE version in BmapiGetFirmwareInfo(). 3. Change the way to show debug message for BMAPI.
6.2.1	7/22/03	Hao-Yang Feng	1. Enhancement BmapiTestLoopBackEx() and BmapiTestLoopBack() to support BMAPI_LOOPBACK_TYPE_EXTERNAL option for 570x based NICs. 2. Add BmapiGetNicStatistics64Ex().
6.2.4	9/8/03	Hao-Yang Feng	Enhance BM_LINK_STATUS_EX.
6.3.0	10/14/03	Hao-Yang Feng	1. Add BmapiGetTeamSnapShot2() 2. Add BmapiTeamFallbackPrimary() 3. Add BmapiGetTeamDrvVersion() 4. Enhance BM_LINK_STATUS_EX
6.3.5	12/18/03	Hao-Yang Feng	Update Appendix section.
6.3.7	1/16/04	Hao-Yang Feng	Update Appendix section.
6.4.0	2/18/04	Hao-Yang Feng	1. Add BmapiGetBRCMNicStatisticsEx() 2. Obsolete BmapiGetTTBRCMNicInfo() and BmapiGetLastDiagPort()
6.4.1	3/3/04	Hao-Yang Feng	Modify BM_ASF_TABLE.
6.4.2	3/12/04	Hao-Yang Feng	Modify BmapiGetPowerMode() and BmapiSetPowerMode().
6.4.8	4/20/04	Hao-Yang Feng	Change to BM_ASF_INFO and BM_ASF_ALRT structures for ASF 2.0-compliance.
6.5.0	5/18/04	Hao-Yang Feng	1. Add BmapiGetBrcmNicParamList(), BmapiGetBrcmNicParamInfo() and BmapiSetBrcmNicParam2(). 2. Obsolete BmapiGetBRCMNicParam(), BmapiSetBRCMNicParam(), BmapiGetMultiBRCMNicParams() and BmapiSetMultiBRCMNicParams()

7.0.1	10/25/04	Hao-Yang Feng	Support for NetXtreme II architecture (5706 series). Due to the large scale of changes, please see each individual API for detail information.
7.1.0	11/12/04	Hao-Yang Feng	Add BmapiGetTeamIDList(), BmapiGetTeamInfo(), BmapiApplyLBFOCfgEx() and BmapiGetTeamStatisticsSnapShotEx().
7.2.6	1/19/05	Hao-Yang Feng	Update 7.1 data structure and definitions.
7.2.7	1/25/05	Hao-Yang Feng	Correct wring Notes for BmapiGetASFTable() and BmapiSetASFTable().
7.2.10	2/17/05	Hao-Yang Feng	Modify definitions for BM_FW_MEDIA_MANUFACT_REGION and BM_NIC_SHARED_CONFIG
7.2.11	2/28/05	Hao-Yang Feng	Add more information for BmapiGetASFTable().
7.3.0	6/9/05	Hao-Yang Feng	Add BmapiGet5706FwInfo(), BmapiTestLEDsAsyncStart() and BmapiTestLEDsAsyncStop().
7.4.0	2/1/06	Hao-Yang Feng	Add BmapiGetASFMailboxCount(), BmapiGetASFMailboxStatus(), BmapiSetASFMailboxStatus(), BmapiGetASFMailboxContents(), BmapiSetASFMailboxContents() and BmapiTestASFMailboxes().
7.5.0	3/8/06	Hao-Yang Feng	Add BmapiTestCable().
7.6.0	11/20/06	Hao-Yang Feng	Add BmapiGetOffloadStackInfo().
7.8.0	9/7/07	Hao-Yang Feng	Add BmapiGetISCSIConfig(), BmapiSetISCSIConfig() and BmapiGet57710FwInfo().
7.9.0	2/27/08	Hao-Yang Feng	Add BmapiGetMgmtProcessors(), BmapiGetMgmtEnableState(), BmapiSetMgmtEnableState(), BmapiAssertMgmtEvent(), BmapiGetMgmtOTPKeys(), BmapiGetMgmtDataLength(), BmapiGetMgmtData(), BmapiSetMgmtData(), BmapiGetMgmtConfigLength(), BmapiGetMgmtConfig(), BmapiSetMgmtConfig(), and BmapiGetMgmtSharedMem().
7.11.0	4/16/08	Hao-Yang Feng	Add BmapiGetISCSIConfig2(), BmapiSetISCSIConfig2(), And BmapiWritePhyFirmware().
7.12.0	6/4/08	Hao-Yang Feng	Add BmapiCreateMgmtData(), BmapiGetMgmtWebDataLength(), BmapiGetMgmtWebData(), BmapiSetMgmtWebData(), BmapiCreateMgmtWebData().

7.13.0	6/16/08	Hao-Yang Feng	Add BmapiRetrieveMultiLinkStatus()
7.14.0	8/29/08	Hao-Yang Feng	Add BmapiGetISCSIRuntimeIPCount(), BmapiGetISCSIRuntimeIP(), BmapiGetISCSIRuntimeStatistics() and BmapiGetISCSISessionStatistics().
7.15.0	10/3/08	Hao-Yang Feng	Add BmapiResetNdisStatistics().
7.16.0	2/18/09	Hao-Yang Feng	Add BmapiWriteFirmware2(), BmapiReadFirmware2(), BmapiGetReverseNWayStatus() and BmapiSetReverseNWay().
7.17.0	5/14/09	Hao-Yang Feng	Add BmapiGetNicStatisticsV3() and BmapiOfldStatistics().
7.19.0	5/7/10	Hao-Yang Feng	Add BmapiGetLldpParams(), BmapiGetDcbxParams(), BmapiGetIscsiCfg(), BmapiSetIscsiCfg(), BmapiGetFcoeCfg(), BmapiSetFcoeCfg(), BmapiGetMBAParams() and BmapiSetMBAParams().

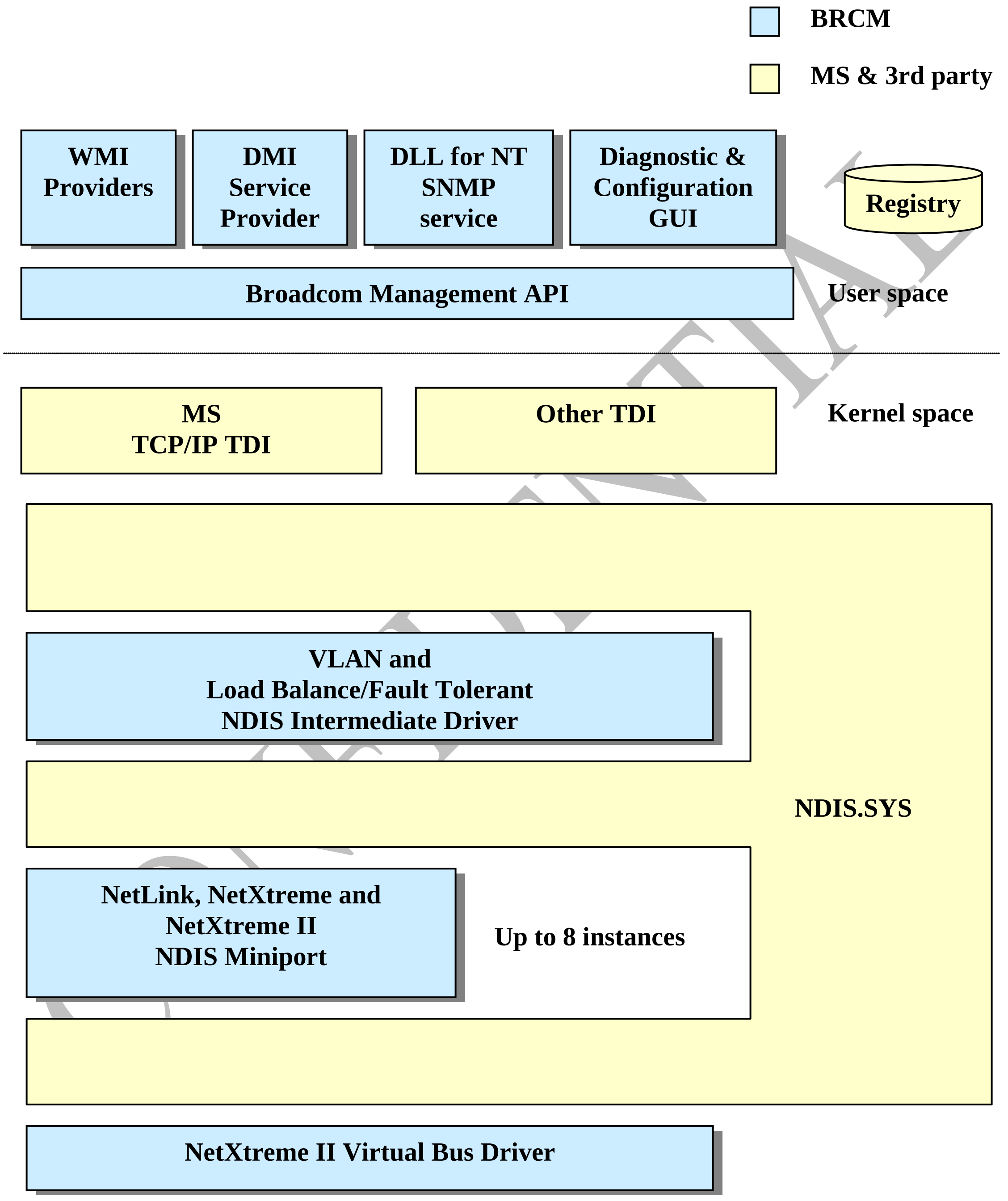
2. Introduction

The purposes to have Broadcom Management API (BMAPI) are

- Help applications to view status, statistics and configuration on network Interface Cards (NIC).
- Help applications to configure and diagnose Broadcom made NIC and configuration related to Broadcom products.
- Help applications not to include driver specific or implementation related knowledge.
- Help applications to immune from the change of driver's implementation.
- Help applications to work with all Broadcom made NIC products with same API set.

Few of BMAPI functions allow applications to set configuration data for Broadcom drivers. However, users **should not** using BMAPI and driver configuration property pages at the same time to configure drivers. The result may be overwritten with each other.

3. Block Diagram



4. Implementation

Broadcom Management API will be implemented in ANSI C as much as possible to make the module more portable. Current implementation supports Windows Me, Windows 98, Microsoft Windows NT 4.0, Windows 2000 and later.

When applications start, applications should:
1. Call BmapiGetVersion() to make sure it is the correct version of BMAPI.
2. Call BmapiInitializeEx() to initialize BMAPI internal data.

When applications end, applications should call BmapiUninitialize() to release internal resources in BMAPI if applications had successfully called BmapiInitialize() or BmapiInitializeEx().

For Windows platforms, applications can enable trace log for BMAPI to help debugging applications. To enable trace log:
1. Delete BMAPI.dll. If any application is currently using and BMAPI.dll is locked, rename BMAPI.dll to something else.
2. Copy BMAPIdbg.dll to the BMAPI.dll directory and rename BMAPIdbg.dll to BMAPI.dll.
3. Restart the system to make sure all applications switch to the new BMAPI dll with debug message.
4. Open registry editor.
5. Open ‘HKEY_LOCAL_MACHINE\SOFTWARE’.
6. Open ‘Broadcom’ key or create one if not exists.
7. Open ‘BMAPI’ key or create one if not exists.
8. Create ‘DebugLevel’ as ‘DWORD’ value.
9. Enter value for ‘DebugLevel’. It could be one or combination of:
   - BMAPI_DEBUG_INFORMATION (0x00000001)
   - BMAPI_DEBUG_WARNING (0x00000002)
   - BMAPI_DEBUG_ERROR (0x00000004)
   - BMAPI_DEBUG_LOG_TO_FILE (0x00000008)

Once ‘DebugLevel’ is set, applications must restart to make the change take effect.

The log file, named ‘BmapiDbg.log’, will be created at the application’s current directory every time application starts if BMAPI_DEBUG_LOG_TO_FILE is set. All trace log messages will be displayed in debugger console if debugger is running.

When trace is enabled, the performance may degrade significantly especially on diagnostics APIs such as BmapiTestCntrolRegisters(), etc.

5. API reference

5.1 BmapiGetVersion

Get version number of BMAPI.

```
void BmapiGetVersion(
OUT U32 *pMajorVersion,
OUT U32 *pMinorVersion,
OUT U32 *pServicePack );
```

Parameters:

`pMajorVersion`

Pointer to major version number.

`pMinorVersion`

Pointer to minor version number.

`pServicePack`

Pointer to service pack number.

Return Value:

No return value.

Supported Network Adapters:

Not applicable

Remarks:

Applications can call this function to verify that it is using the correct version of BMAPI. Applications can call this function at any time i.e. it can be called before BMAPI is initialized.

5.2 BmapiInitialize

Initialize internal data and read NIC configuration from system.

```
U32 BmapiInitialize( void );
```

Parameters:

This function does not have parameters.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications **MUST** call this function before calling any other functions except BmapiGetVersion(). Applications only need to call once. However, application can call more than once without getting error. When applications are finished, applications **MUST** call BmapiUninitialize().
The calling thread of BmapiUninitialize() **MUST** be the thread called BmapiInitialize().
Multithread applications should call BmapiInitializeEx().

5.3 BmapiUninitialize

Release allocated resources.

```
U32 BmapiUninitialize( void );
```

Parameters:

This function does not have parameters.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications **MUST** call BmapiUninitialize() before exiting, otherwise, application may have memory leak. An application need to call BmapiUninitialize() as many times as it calls BmapiInitialize() or BmapiInitializeEx().
The calling thread of BmapiUninitialize() **MUST** be the thread called BmapiInitialize().

5.4 BmapiGetNumPhyNic

The BmapiGetNumPhyNic will return the number of physical network adapters that are installed with drivers in the system.

```
U32 BmapiGetNumPhyNic(
  OUT U32 *pNumOfPhy );
```

Parameters:

`pNumOfPhy`

Pointer to the number of physical network adapters.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including 4401, 57xx, NetXtreme II and third party NICs

Remarks:

Applications call this function to understand how many physical network adapters are installed with driver in the system, allocate enough buffers to fill all adapters' data and call BmapiGetAllPhyNic() to retrieve all data.

5.5 BmapiGetAllPhyNic

The BmapiGetAllPhyNic will return information for all physical network adapters.

```
U32 BmapiGetAllPhyNic(
  OUT BM_ADAPTER_INFO *pPhyList,
  IN U32 uNumOfPhy );
```

Parameters:

`pPhyList`

Pointer to the array of BM_ADAPTER_INFO for all physical network adapters.

`uNumOfPhy`

Number of BM_ADAPTER_INFO structure allocated for *pPhyList* array. This number must be more than the number of physical network adapters in the system. If not, the function will return error code BMAPI_BUFSHORT.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including 4401, 57xx, NetXtreme II and third party NICs

Remarks:

Applications must call BmapiGetNumPhyNic() to understand how many NICs are in system first. Applications, then, allocate enough buffers to fill all adapters' data and call BmapiGetAllPhyNic() to retrieve data. If uNumOfPhy is not enough, the function will return error code.

5.6 BmapiGetNumUnassignedNic

The BmapiGetNumUnassignedNic will return the number of physical network adapters that are not part of any BASP team (Load Balance/Fail Over team) in the system.

```
U32 BmapiGetNumUnassignedNic(
  OUT U32 *pNumOfUnassigned );
```

Parameters:

`pNumOfUnassigned`

Pointer to the number of unassigned network adapters.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including 4401, 57xx, NetXtreme II L2 NDIS and third party NICs

Remarks:

Applications call this function to understand how many unassigned network adapters are in the system, allocate enough buffers to fill all adapters’ data and call BmapiGetAllUnassignedNic() to retrieve all data.

5.7 BmapiGetAllUnassignedNic

The BmapiGetAllUnassignedNic will return information for all physical network adapters that are not assigned to any BASP team.

```
U32 BmapiGetAllUnassignedNic(
  OUT BM_ADAPTER_INFO *pUnassignedList,
  IN U32 uNumOfUnassigned );
```

Parameters:

`pUnassignedList`

Pointer to the array of BM_ADAPTER_INFO for all unassigned network adapters.

`uNumOfUnassigned`

Number of adapter structure allocated for *pUnassignedList* array. This number must be more than the number of unassigned network adapters in the system. If not, the function will return error code BMAPI_BUFSHORT.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including 4401, 57xx, NetXtreme II L2 NDIS and third party NICs

Remarks:

Applications must call BmapiGetNumUnassignedNic() to understand how many unassigned network adapters are in system first. Applications, then, allocate enough buffers to fill all adapters' data and call BmapiGetAllUnassignedNic() to retrieve data. If uNumOfUnassigned is not enough, the function will return error code.

5.8 BmapiGetNumTeam

The BmapiGetNumTeam will return the number of BASP team (Load Balance/Fail Over team) configured in the system.

```
U32 BmapiGetNumTeam(
  OUT U32 *pNumOfTeam );
```

Parameters:

`pNumOfTeam`

Pointer to the number of team.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications call this function to understand how many BASP teams are configured in system, allocate enough buffers to fill all teams' data and call BmapiGetAllTeam() to retrieve all data.

5.9 BmapiGetAllTeam

The BmapiGetAllTeam will return information for all BASP teams.

```
U32 BmapiGetAllTeam(
  OUT BM_TEAM_INFO *pTeamList,
  IN U32 uNumOfTeam );
```

Parameters:

pTeamList

Pointer to the array of BM_TEAM_INFO for all BASP teams.

uNumOfTeam

Number of team structure allocated for *pTeamList* array. This number must be more than the number of teams in the system. If not, the function will return error code BMAPI_BUFSHORT.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications must call BmapiGetNumTeam() to understand how many teams are configured in system. Applications, then, allocate enough buffers to fill all teams' data and call BmapiGetAllTeam() to retrieve data. If uNumOfTeam is not enough, the function will return error code.

5.10 BmapiApplyLBFOCfg

The BmapiApplyLBFOCfg will apply teaming changes for Broadcom Advanced Server Program (load balancing, fail over and VLAN).

```
U32 BmapiApplyLBFOCfg(
  IN BM_TEAM_INFO *pTeamList,
  IN U32 uNumOfTeam,
  OUT U32 *pReboot );
```

Parameters:

pTeamList

Pointer to the array of BM_TEAM_INFO for all teams.

uNumOfTeam

Number of team structure allocated for *pTeamList* array.

pReboot

Pointer to a boolean flag to indicate system need to reboot or not.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

4401, 57xx, NetXtreme II L2 NDIS and third party NICs

Remarks:

The function may trigger binding if it is necessary.
If all teams are deleted, pass pTeamList as NULL and uNumOfTeam as 0.
If 'pReboot' is non-zero on function return, reboot is required.
The function will re-initialize BMAPI internal data to reflect the latest configuration.
Application should always read all physical network adapters, virtual network adapters and team information after calling this function regardless of the result of the function.

Users **should not** use BMAPI and driver configuration property pages at the same time to configure drivers. The result may be overwritten with each other.

5.11 BmapiGetTeamListSnapShot

The BmapiGetTeamListSnapShot will return to caller a list of team IDs that are configured and running in teaming (VLAN/Load Balance/Fail Over) driver.

```
U32 BmapiGetTeamListSnapShot(
  IN OUT U32 *pNumOfTeam,
  OUT U32 *pIDList );
```

Parameters:

`pNumOfTeam`

On input, pointer to number of U32 allocated in *pIDList* array. On output, number of team IDs in *pIDList* array.

`pIDList`

Pointer to team ID array.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications call this function to understand the number of teams in driver and their corresponding team ID. Application can use the team ID to call BmapiGetTeamSnapShot() to retrieve a team's data from driver.
If pIDList is NULL, pNumOfTeam will be the number of teams in driver. Applications can call the function with `pIDList` set to NULL first to get the number of team. Then, call the function second time with enough buffer allocated and assigned to `pIDList` to retrieve the id list.

5.12 BmapiGetTeamSnapShot

The BmapiGetTeamSnapShot will return information for a team from teaming driver.

```
U32 BmapiGetTeamSnapShot(
  IN U32 teamId,
  OUT BM_TEAM_INFO *pTeam );
```

Parameters:

`teamId`

Team ID of the team information application wants to retrieve.

`pTeam`

Pointer to a BM_TEAM_INFO structure for information of a team.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications must call BmapiGetTeamListSnapShot() to learn available teams information in teaming driver. Applications, then, call BmapiGetTeamSnapShot() to retrieve team information.

5.13 BmapiGetTeamStatisticsSnapShot

The BmapiGetTeamStatisticsSnapShot will return statistics information for a team from teaming driver.

```
U32 BmapiGetTeamStatisticsSnapShot(
  IN U32 teamId,
  OUT BM_TEAM_STATISTICS *pTeamStatistics );
```

Parameters:

`teamId`

Team ID of the team statistics information application wants to retrieve.

`pTeamStatistics`

Pointer to a BM_TEAM_STATISTICS structure for statistics information of a team.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications must call BmapiGetTeamListSnapShot() to learn available teams information in teaming driver. Applications, then, call BmapiGetTeamStatisticsSnapShot() to retrieve statistics information of a team.

5.14 BmapiDoNicIOCTL

The BmapiDoNicIOCTL will issue IOCTL command to the driver that the ‘handle’ belongs to.

```
U32 BmapiDoNicIOCTL(
  IN U32 handle,
  IN U32 ioctlCode,
  IN void *pInBuf,
  IN U32 inBufSize,
  IN/OUT void *pOutBuf,
  IN/OUT U32 *pOutBufSize,
  OUT U32 *pResult );
```

Parameters:

`handle`

Handle to an adapter. (can be found in BM_ADAPTER_INFO structure)

`ioctlCode`

IOCTL control code.

`pInBuf`

Pointer to an input buffer that will pass to driver.

`inBufSize`

Size of the input buffer pointed *pInBuf*.

`pOutBuf`

Pointer to an output buffer that will pass to driver and filled by driver.

`pOutBufSize`

On input, it is the size of the output buffer pointed *pOutBuf*. On output, it is the amount of data stored in *pOutBuf*.

`pResult`

Applications must pass a pointer to a U32 variable. The variable will store the result of GetLastError().

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including 4401, 57xx, NetXtreme II and its virtual devices and third party NICs

Remarks:

Applications could call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO. Applications should use the 'handle' to call BmapiDoNicIOCTL(). If the driver is not running at the time application calls BmapiDoNicIOCTL(), BmapiDoNicIOCTL() will return error code.

5.15 BmapiGetNicStatistics

The BmapiGetNicStatistics will return statistics information for a network adapter.

```
U32 BmapiGetNicStatistics(
  IN U32 handle,
  OUT BM_GENERAL_STATISTICS *pGenStatistics,
  OUT BM_ETHERNET_STATISTICS *pEthStatistics );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pGenStatistics`

Pointer to a `BM_GENERAL_STATISTICS` structure for general statistics information of a network adapter. The data will be filled out on return.

`pEthStatistics`

Pointer to an `BM_ETHERNET_STATISTICS` structure for ethernet statistics information of a network adapter. The data will be filled out on return.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including 4401, 57xx, NetXtreme II and its L2 NDIS and third party NICs

Remarks:

Applications should allocate buffers for the statistics structures and pass pointers to those structures to the function. Application should be aware that some information might not be available. In case of NIC is disabled, the return code will be BMAPI_DRIVER_NOT_LOADED.

5.16 BmapiRetrieveLinkStatus

The BmapiRetrieveLinkStatus will retrieve link status and link negotiation status for Broadcom made network adapters.

```
U32 BmapiRetrieveLinkStatus(
  IN U32 handle,
  OUT BM_LINK_STATUS *pLinkStatus );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pLinkStatus`

On input, applications pass a pointer to a BM_LINK_STATUS structure. On output, data will be filled out in the BM_LINK_STATUS structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including 4401, 57xx, NetXtreme II and it's L2 NDIS and third party NICs. All BASP created virtual adapters.

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
For all non-Broadcom made network adapters and BASP created virtual adapters, only "link_status" is available.
For 4401 network adapters, only "link_status", "duplex_mode" and "link_speed" are available.

5.17 BmapiGetBRCMNicInfo

The BmapiGetBRCMNicInfo will retrieve proprietary Broadcom network adapter information.

```
U32 BmapiGetBRCMNicInfo(
  IN U32 handle,
  OUT BM_BRCM_ADAPTER_INFO *pBRCMNicInfo );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pBRCMNicInfo

On input, applications pass a pointer to a BM_BRCM_ADAPTER_INFO structure. On output, data will be filled out in the BM_BRCM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

57xx, 4401, NetXtreme II and its L2 NDIS

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.18 BmapiTestControlRegisters

The BmapiTestControlRegisters will return test result of control registers for Broadcom made network adapters.

```
BOOL BmapiTestControlRegisters(
  IN U32 handle,
  IN OUT CHAR *pRstr );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pRstr

On input, applications need to pass a 200 bytes long buffer. On output, if the function returns error, BMAPI may return an ASCII string to indicate the reason of error.

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
The function will not validate the buffer. If applications do not pass a correct length of buffer, memory corruption may occur.
Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC.
After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.

5.19 BmapiTestMIIRegisters

The BmapiTestMIIRegisters will return test result of MII control registers for Broadcom made network adapters.

```
BOOL BmapiTestMIIRegisters(
  IN U32 handle,
  IN OUT CHAR *pRstr );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pRstr`

On input, applications need to pass a 200 bytes long buffer. On output, if the function returns error, BMAPI may return an ASCII string to indicate the reason of error.

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
The function will not validate the buffer. If applications do not pass a correct length of buffer, memory corruption may occur.

Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC. After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.

5.20 BmapiTestEEPROM

The BmapiTestEEPROM will return test result of EEPROM for Broadcom made network adapters.

```
BOOL BmapiTestEEPROM(
  IN U32 handle,
  IN OUT CHAR *pRstr );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pRstr`

On input, applications need to pass a 200 bytes long buffer. On output, if the function returns error, BMAPI may return an ASCII string to indicate the reason of error.

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.

The function will not validate the buffer. If applications do not pass a correct length of buffer, memory corruption may occur.

Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup. After the function returned, call BmapiUnInitDiag() to terminate diagnostic setup.

5.21 BmapiTestInternalMemory

The BmapiTestInternalMemory will test internal memory and return result for Broadcom made network adapters.

```
BOOL BmapiTestInternalMemory(
  IN U32 handle,
  IN OUT CHAR *pRstr );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pRstr`

On input, applications need to pass a 200 bytes long buffer. On output, if the function returns error, BMAPI may return an ASCII string to indicate the reason of error.

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

The function will not validate the buffer. If applications do not pass a correct length of buffer, memory corruption may occur.

Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC.

After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.

5.22 BmapiTestInterrupt

The BmapiTestInterrupt will test internal memory and return result for Broadcom made network adapters.

```
BOOL BmapiTestInterrupt(
  IN U32 handle,
  IN OUT CHAR *pRstr );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pRstr`

On input, applications need to pass a 200 bytes long buffer. On output, if the function returns error, BMAPI may return an ASCII string to indicate the reason of error.

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
The function will not validate the buffer. If applications do not pass a correct length of buffer, memory corruption may occur.
Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC.
After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.

5.23 BmapiTestLoopBack

The BmapiTestLoopBack will perform loopback test and return test result for Broadcom made network adapters.

```
BOOL BmapiTestLoopBack(
  IN U32 handle,
  IN ULONG Lbtype,
  IN OUT CHAR *pRstr );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

Lbtype

Loopback type to perform. Can be BMAPI_LOOPBACK_TYPE_MAC, BMAPI_LOOPBACK_TYPE_PHY or BMAPI_LOOPBACK_TYPE_EXTERNAL.

pRstr

On input, applications need to pass a 200 bytes long buffer. On output, if the function returns error, BMAPI may return an ASCII string to indicate the reason of error.

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
The function will not validate the buffer. If applications do not pass a correct length of buffer, memory corruption may occur.
Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup. After the function returned, call BmapiUnInitDiag() to terminate diagnostic setup.
Before calling this function, applications MUST make sure miniport driver is NOT in suspend mode. If it does, applications need to call BmapiResumeDriver() to get miniport driver running.

5.24 BmapiTestCPU

The BmapiTestCPU will enable test mode and test the on chip processor for Broadcom made network adapters.

```
BOOL BmapiTestCPU(
  IN U32 handle,
  IN OUT CHAR *pRstr );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pRstr

On input, applications need to pass a 200 bytes long buffer. On output, if the function returns error, BMAPI may return an ASCII string to indicate the reason of error.

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
The function will not validate the buffer. If applications do not pass a correct length of buffer, memory corruption may occur.
Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC. After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.

5.25 BmapiTestLEDs

The BmapiTestLEDs will flip the LEDs to indicate which logical NIC card is bound to physical NIC card for Broadcom made network adapters.

```
BOOL BmapiTestCPU(
  IN U32 handle,
  IN ULONG BlinkDurationSec );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`BlinkDurationSec`

Specify how long to flip the LEDs (in seconds, up to 120 seconds).

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup. After the function returned, call BmapiUnInitDiag() to terminate diagnostic setup.

5.26 BmapiGetServiceName

The BmapiGetServiceName will return the service name corresponding to the adapter's handle.

```
U32 BmapiGetServiceName(
  IN U32 handle,
  OUT U8 *pServiceName,
  IN U32 uBufLen );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pServiceName`

Pointer to the buffer that will be filled with the adapter's service name on function's return.

`uBufLen`

Length of buffer pointed by 'pServiceName'.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including 4401, 57xx, NetXtreme II and its L2 NDIS and third party NICs. All BASP created virtual adapters.

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.27 BmapiGetBRCMNicStatistics

The BmapiGetBRCMNicStatistics will return Broadcom network adapters proprietary statistics data.

```
U32 BmapiGetBRCMNicStatistics(
  IN U32 handle,
  IN/OUT BM_BRCM_STATISTICS *pBrcmStatistics );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pBrcmStatistics`

Pointer to a BM_BRCM_STATISTICS structure for Broadcom network adapter proprietary statistics information. The data will be filled out on return.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II and its L2 NDIS

Remarks:

Applications should allocate buffers for the statistics structures and pass pointers to those structures to the function. In case of NIC is disabled, the return code will be BMAPI_DRIVER_NOT_LOADED.

5.28 BmapiForceBRCMNicLinkSpeed

The BmapiForceBRCMNicLinkSpeed will force the Broadcom miniport driver to set the link speed to the specified link speed.

```
U32 BmapiForceBRCMNicLinkSpeed(
  IN U32 handle,
  IN U32 uAutoNegotiation,
  IN U32 uLinkSpeed );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

uAutoNegotiation

Using auto-negotiation advertisement or force the setting.

uLinkSpeed

The forced speed to set to. Possible values include

BMAPI_LM_REQUESTED_MEDIA_TYPE_AUTO (Not available if 'uAutoNegotiation' is false)
BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_10MBPS
BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_10MBPS_FULL_DUPLEX
BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_100MBPS
BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_100MBPS_FULL_DUPLEX
BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_1000MBPS
BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_1000MBPS_FULL_DUPLEX (Not available if 'uAutoNegotiation' is false)

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II (5706 family only)

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
4401 is supported on Windows 2000 or later.

5.29 BmapiLBFOSoftwareStatus

BmapiLBFOSoftwareStatus will test Load Balance/Fault Tolerance driver status.

```
U32 BmapiForceBRCMNicLinkSpeed(
  OUT U32 *pStatus );
```

Parameters:

pStatus

Pointer to status of Load Balance/Fault Tolerance. The value could be BMAPI_LBFO_STATUS_NOT_INSTALLED, BMAPI_LBFO_STATUS_NOT_LOADED or BMAPI_LBFO_STATUS_LOADED.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications can call this function to understand the status of Load Balance/Fault Tolerance software.

5.30 BmapiGetPnpDevId

BmapiGetPnpDevId will retrieve PnP device ID for a physical network adapter in Windows 98, Windows Me and Windows 2000 or later.

```
U32 BmapiGetPnpDevId(
  IN U32 handle,
  OUT U8 *pDevID,
  IN OUT U32 *pBufLen,
  OUT U32 *pRegIndex,
  OUT U32 *pSlotNumber );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pDevID

Pointer to output buffer that will be filled with PnP device ID if function returns BMAPI_OK.

pBufLen

On input, pointer to the length of buffer passed in. On output, the function will fill it with the length of buffer used.

pRegIndex

Index retrieved from registry key. For Example,
"{4D36E972-E325-11CE-BFC1-08002BE10318}\[num]"
The 'num' starts from 0000. pRegIndex will get the value 'num' and convert it into U32 type.
This is ignored For Windows 98 and Windows Me.

pSlotNumber

Specifies a number associated with the device that can be displayed in the user interface. This number is typically a user-perceived slot number, such as a number printed next to the slot on the board, or some other number that makes locating the physical device easier for the user. For buses with no such convention, or when the ‘UINumber’ is unknown, the bus driver leaves this at its default value of 0xFFFFFFFF.
This is ignored For Windows 98 and Windows Me.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtreme I, NetXtreme II and its L2 NDIS and third party NICs. All BASP created virtual adapters.

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
If buffer is too short, the function will return BMAPI_BUFSHORT and `pBufLen` will be filled with the length required.
The API supports Windows 98, Me, 2000 and later. NT 4 is not supported.
On Windows 98 and Me, only “DevID” is available.
The form of PnP device ID is <enumerator>\<Device ID>\<Instance ID>.
If *pBufLen == 0, function will return with required buffer length.
'pRegIndex' and 'pSlotNumber' are optional. If 'pRegIndex' is not NULL and no driver is installed for the device, error code BMAPI_DRIVER_NOT_INSTALLED will be returned.

5.31 BmapiSuspendDriver

BmapiSuspendDriver will suspend Broadcom miniport driver and bring down the link (as cable disconnected).

```
BOOL BmapiSuspendDriver(
  IN U32 handle,
  IN OUT CHAR *pRstr );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pRstr`

On input, applications need to pass a 200 bytes long buffer. On output, if the function returns error, BMAPI may return an ASCII string to indicate the reason of error.

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
The function will not validate the buffer. If applications do not pass a correct length of buffer, memory corruption may occur.

5.32 BmapiResumeDriver

BmapiResumeDriver will resume Broadcom miniport driver from suspend mode.

```
BOOL BmapiResumeDriver(
  IN U32 handle,
  IN OUT CHAR *pRstr );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pRstr`

On input, applications need to pass a 200 bytes long buffer. On output, if the function returns error, BMAPI may return an ASCII string to indicate the reason of error.

Return Value:

Return TRUE if successful; otherwise return FALSE.

Supported Network Adapters:

NetLink, NetXtreme I

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
The function will not validate the buffer. If applications do not pass a correct length of buffer, memory corruption may occur.

5.33 BmapiEnableDevice

BmapiEnableDevice will enable or disable a network adapter in Windows 2000 or later.

```
U32 BmapiEnableDevice(
  IN U32 handle,
  IN U32 uEnable );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

uEnable

Non-zero value to enable the device, zero value to disable the device.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtreme I and NetXtreme I and its virtual devices and third party NICs. All BASP created virtual adapters.

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.

5.34 BmapiRegisterEvent

The function will register the event to BMAPI and BMAPI will call back to applications when the event occurred.

```
U32 BmapiRegisterEvent(
  IN U32 event,
  IN BMAPIEVENTCALLBACK func,
  IN void *cookie,
  OUT U32 *pEeventHandle );
```

Parameters:

event

The event BMAPI will call back.

func

The function to call back to.

cookie

A pointer reserved for calling function. The pointer is not used by BMAPI and it will be passed back in the call back function. Applications can use the pointer to pass parameters to its own call back function.

pEventHandle

This is a pointer to the handle that will be filled by BMAPI on return. The handle is to identify the registration of the event. Applications need to use the handle to unregister the event.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications must call BmapiInitialize()/BmapiInitializeEx() before calling this function.

If the team ID indicated by an event can not be found in registry, the team name, adapter service name and adapter handle may not be passed to callback function.

In the case of event BMAPI_EVT_ADD and BMAPI_EVT_REMOVE, application should refresh BMAPI internal data by calling BmapiRefreshData(). **(Applications should NEVER call BMAPI APIs from the call back thread.)** In the case of Hot Plug machine, if applications failed to do so, BMAPI will not be able to provide detail information for the event. However, **if applications are calling BmapiApplyLBFOCfg(), application do not need to call BmapiRefreshData()** because the BmapiApplyLBFOCfg() will call BmapiRefreshData() before return.

In the callback function, generally, applications will get information about which team and which adapter change its status and other related information. However, if applications did not refresh BMAPI as described in previous paragraph, BMAPI may not be able to provide detail information to applications. In fact, application will only get correct team ID, number of active load balance member and number of stand by member and applications will not know which adapter causing the event.

Applications should not spend too much time in the call back function because it will block the calling thread in BMAPI and prevent it to deliver events to next callback functions if there are multiple call back functions registered for the same event from the same process.

In the case of event BMAPI_EVT_ACTIVE and BMAPI_EVT_INACTIVE, applications could use 'extra_info' from the callback function parameters to understand the cause of the active or inactive event.

5.35 BmapiUnRegisterEvent

The function will unregister the event for callback.

```
U32 BmapiUnRegisterEvent(
  IN U32 uEeventHandle );
```

Parameters:

uEventHandle

The event to unregister.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

none

5.36 BmapiInitializeEx

Initialization function of BMAPI and applications can have a choice to decide whether to keep COM initialized or not. If not, COM will be uninitialized immediately before the function return.

```
U32 BmapiInitializeEx(
  IN U32 bKeepCom );
```

Parameters:

`bKeepCom`

Keep COM initialized or not.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications **MUST** call this function before calling any other functions except BmapiGetVersion(). Applications only need to call once. However, application can call more than once without getting error. After calling BmapiInitializeEx(), applications do not need to call BmapiInitialize().

Applications **MUST** call BmapiUninitialize() before exiting, otherwise, application may have memory leak.

Applications that call BmapiInitializeEx(TRUE) MUST use the same thread to call BmapiUninitialize().

Multithread applications that call BmapiInitializeEx() and BmapiUninitialize() from different threads must use BmapiInitializeEx(FALSE).

5.37 BmapiIsInitialized

Applications use the function to determine BMAPI is initialized or not.

```
U32 BmapiIsInitialized()
```

Parameters:

none

Return Value:

Return BMAPI_OK if BMAPI is initialized; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications can call this function without call BmapiInitialize() or BmapiInitializeEx(). This function will work well with single thread applications or applications that can control the calls to BmapiInitialize()/BmapiInitializeEx(). However, if the application is multiple threaded and the API calls to BmapiInitialize()/BmapiInitializeEx() are scattered and not controllable, the function will not guarantee correct result.

5.38 BmapiGetBRCMNicParam

The API is obsolete.

5.39 BmapiSetBRCMNicParam

The API is obsolete.

5.40 BmapiGetMultiBRCMNicParams

The API is obsolete.

5.41 BmapiSetMultiBRCMNicParams

The API is obsolete.

5.42 BmapiRefreshData

BmapiRefreshData() will force BMAPI to refresh its internal data.

```
U32 BmapiRefreshData();
```

Parameters:

none

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

For multithreaded applications, once a thread calls the function, all other threads will see the change of data.

5.43 BmapiGetHandleByServiceName

The BmapiGetHandleByServiceName will return the adapter's handle corresponding to the adapter's service name.

```
U32 BmapiGetHandleByServiceName(
  IN U8 *pServiceName,
  OUT U32 *handle );
```

Parameters:

pServiceName

Pointer to the ASCII NULL terminated service name.

handle

Pointer to a handle for an adapter.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtreme I, NetXtreme II and its L2 NDIS and third party NICs. All BASP created virtual adapters.

Remarks:

Make sure BMAPI is initialized before calling the function. If the NIC is not found, return code is BMAPI_NIC_NOT_FOUND.

5.44 BmapiIMOnlineDevice

The BmapiIMOnlineDevice will ask intermediate driver to pass or stop passing traffic to the miniport.

```
U32 BmapiIMOnlineDevice(
  IN U32 team_id,
  IN U32 handle,
  IN U32 bOnline );
```

Parameters:

`team_id`

Team ID of the miniport joined.

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`bOnline`

Flag to indicate to pass traffic to the miniport or not.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.45 BmapiGetNicStatistics64

The BmapiGetNicStatistics64 will return all 64-bit statistics information for a network adapter.

```
U32 BmapiGetNicStatistics64(
  IN U32 handle,
  OUT BM_GENERAL_STATISTICS64 *pGenStatistics,
  OUT BM_ETHERNET_STATISTICS64 *pEthStatistics );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pGenStatistics`

Pointer to a `BM_GENERAL_STATISTICS64` structure for general statistics information of a network adapter. The data will be filled out on return.

`pEthStatistics`

Pointer to an `BM_ETHERNET_STATISTICS64` structure for ethernet statistics information of a network adapter. The data will be filled out on return.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtremeI and NetXtreme II and its L2 NDIS and third party NICs.

Remarks:

Applications should allocate buffers for the statistics structures and pass pointers to those structures to the function. Application should be aware that some information might not be available. In case of NIC is disabled, the return code will be BMAPI_DRIVER_NOT_LOADED.

5.46 BmapiGetNicPciInfo

The BmapiGetNicPciInfo will return PCI information of the NIC.

```
U32 BmapiGetNicPciInfo(
  IN U32 handle,
  OUT BM_NIC_PCI_INFO *pNicPciInfo );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pNicPciInfo`

Pointer to the buffer that will be filled with the adapter's PCI information. 'version' field MUST be filled on input.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtremeI and NetXtreme II and its L2 NDIS and third party NICs.

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function. If PCI IDs can not be determined for the NIC (for example, NIC is removed from the system under NT 4.0), PCI IDs will be 0xffff.

Applications **MUST** set ‘version’ field in BM_NIC_PCI_INFO structure prior to call the function.

5.47 BmapiInitDiag

The BmapiInitDiag will lock the access to a NIC during diagnostic so that no multiple diagnostics can perform on the same NIC concurrently.

```
U32 BmapiInitDiag(
  IN U32 handle );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtremeI and NetXtreme II

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

Applications **MUST** call this API prior to do any NIC diagnostic test including cable analysis. Otherwise, the test will fail. Applications also **MUST** call BmapiUnInitDiag() when all diagnostic tasks are done.

5.48 BmapiUnInitDiag

The BmapiUnInitDiag unlock the access to a NIC locked by BmapiInitDiag() API.

```
U32 BmapiUnInitDiag(
  IN U32 handle );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtremeI and NetXtreme II

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.49 BmapiGetNumPhyNicEx

The function will return the number of physical network adapters in the system. In Windows 2000 or above, the number includes network adapters sitting in PCI slots but not installed with drivers.

```
U32 BmapiGetNumPhyNicEx(
  IN U32 *pNumOfPhy );
```

Parameters:

pNumOfPhy

Pointer to the number of physical network adapters.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtremeI, NetXtreme II and third party NICs.

Remarks:

Applications call BmapiGetNumPhyNicEx() to understand how many physical network adapters are in the system, allocate enough buffers to fill all adapters' handles, call BmapiGetAllPhyNicHandles() to retrieve all handles and call BmapiGetPhyNic() to retrieve a specific network adapter’s info.

5.50 BmapiGetAllPhyNicHandles

The function will return handles for all physical network adapters.

```
U32 BmapiGetAllPhyNicHandles(
  OUT U32 *pHandleList,
  IN U32 uNumOfPhy );
```

Parameters:

pHandleList

Pointer to the array of physical network adapter handles.

uNumOfPhy

Number of handles allocated in *`pHandleList`* array.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtremeI, NetXtreme II and third party NICs.

Remarks:

Applications call BmapiGetNumPhyNicEx() to understand how many physical network adapters are in the system, allocate enough buffers to fill all adapters' handles, call BmapiGetAllPhyNicHandles() to retrieve all handles and call BmapiGetPhyNic() to retrieve a specific network adapter’s info.

5.51 BmapiGetPhyNic

The function will return the physical network adapter information according the handle.

```
U32 BmapiGetPhyNic(
  IN U32 handle,
  OUT BM_ADAPTER_INFO_EX *pNicInfoEx );
```

Parameters:

`handle`

Handle to the NIC information that should be retrieved.

`pNicInfoEx`

Pointer to the buffer for physical network adapter information.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtremeI, NetXtreme II and its virtual devices and third party NICs.

Remarks:

Applications call BmapiGetNumPhyNicEx() to understand how many physical network adapters are in the system, allocate enough buffers to fill all adapters' handles, call BmapiGetAllPhyNicHandles() to retrieve all handles and call BmapiGetPhyNic() to retrieve a specific network adapter’s info.
Make sure set the ‘version’ in BM_ADAPTER_INFO_EX to BMAPI_ADAPTER_INFO_EX_VER before calling the function.

5.52 BmapiGetASFTable

BmapiGetASFTable will read ASF table from a Broadcom made ASF capable network adapter.

```
U32 BmapiGetASFTable(
  IN U32 handle,
  OUT BM_ASF_TABLE *pNicASF );
```

Parameters:

handle

Handle to an adapter.

pNicASF

Application allocated buffer for ASF table.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetXtreme I (exclude 5700 and 5701) with ASF firmware

Remarks:

Applications must call BmapiGetAllPhyNic(), BmapiGetAllPhyNicHandles etc. to get adapter handle.
Make sure set the ‘version’ in BM_ASF_TABLE to BMAPI_ASF_T_VERSION before calling the function.
If the application had called BmapiInitDiag() prior to call this API, BmapiInitDiag() must be called from the same thread that calls this API.
The function can be used to read configuration for both ASF and IPMI. If the management firmware is IPMI, the ‘flags’ defined in BM_ASF_TABLE will have BM_IPMI_SUPPORT set. The same ‘AsfGui.cfg.Config.EnableASF’ flag can be used to determine whether the firmware is enabled or not.
The function supports UMP firmware. If the firmware is UMP, the ‘flags’ defined in BM_ASF_TABLE will have BM_UMP_SUPPORT set. All ASF and IPMI configuration information will no be populated for UMP. Please refer to BM_ASF_TABLE in BMAPI.h for detail information.

5.53 BmapiSetASFTable

BmapiSetASFTable will write ASF table to a Broadcom made ASF capable network adapter.

```
U32 BmapiGetASFTable(
  IN U32 handle,
  OUT BM_ASF_TABLE *pNicASF );
```

Parameters:

`handle`

Handle to an adapter.

`pNicASF`

ASF table to write to the network adapter.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetXtreme I (exclude 5700 and 5701) with ASF firmware

Remarks:

Applications must call BmapiGetAllPhyNic(), BmapiGetAllPhyNicHandles etc. to get adapter handle.
This API will suspend and resume driver at the end of NVRAM access.
Make sure set the ‘version’ in BM_ASF_TABLE to BMAPI_ASF_T_VERSION before calling the function.
If the application had called BmapiInitDiag() prior to call this API, BmapiInitDiag() must be called from the same thread that calls this API.

5.54 BmapiGetBIOS

BmapiGetBIOS will read BIOS information from the 'stat_addr' for 'Length' bytes long.

```
U32 BmapiGetBIOS(
  IN U64 start_addr,
  OUT void *buffer,
  IN U32 length );
```

Parameters:

`start_addr`

Starting address of physical memory to read.

`buffer`

Application allocated buffer to store data read from the physical memory.

`length`

bytes to read and length of *`buffer`*

Return Value:
Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:
Not applicable

Remarks:
Make sure BMAPI is initialized before calling this function..

5.55 BmapiTestControlRegistersEx

The BmapiTestControlRegistersEx will return test result of control registers for Broadcom made network adapters.

```
U32 BmapiTestControlRegistersEx(
    IN U32 handle );
```

Parameters:
`handle`
Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:
Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:
NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:
Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC.
After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.

5.56 BmapiTestMIIRegistersEx

The BmapiTestMIIRegistersEx will return test result of MII control registers for Broadcom made network adapters.

```
U32 BmapiTestMIIRegistersEx(
    IN U32 handle );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC.
After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.

5.57 BmapiTestEEPROMEx

The BmapiTestEEPROMEx will return test result of EEPROM for Broadcom made network adapters.

```
U32 BmapiTestEEPROMEx(
    IN U32 handle );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup. After the function returned, call BmapiUnInitDiag() to terminate diagnostic setup.

5.58 BmapiTestInternalMemoryEx

The BmapiTestInternalMemoryEx will test internal memory and return result for Broadcom made network adapters.

```
U32 BmapiTestInternalMemoryEx(
    IN U32 handle );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC. After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.

5.59 BmapiTestInterruptEx

The BmapiTestInterruptEx will test internal memory and return result for Broadcom made network adapters.

```
U32 BmapiTestInterruptEx(
    IN U32 handle );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.

Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC. After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.

5.60 BmapiTestLoopBackEx

The BmapiTestLoopBackEx will perform loopback test and return test result for Broadcom made network adapters.

```
U32 BmapiTestLoopBackEx(
    IN U32 handle,
    IN ULONG Lbtype );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

Lbtype

Loopback type to perform. Can be BMAPI_LOOPBACK_TYPE_MAC, BMAPI_LOOPBACK_TYPE_PHY or BMAPI_LOOPBACK_TYPE_EXTERNAL.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.

Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup. After the function returned, call BmapiUnInitDiag() to terminate diagnostic setup.

For 4401 and 57xx NICs, before calling this function, applications MUST make sure miniport driver is NOT in suspend mode. If it does, applications need to call BmapiResumeDriver() to get miniport driver running.

For 5706 family, before calling the function, make sure application had called BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC. BMAPI_LOOPBACK_TYPE_EXTERNAL is upported for 570x based NICs only.

5.61 BmapiTestCPUEx

The BmapiTestCPUEx will enable test mode and test the on chip processor for Broadcom made network adapters.

```
U32 BmapiTestCPUEx(
    IN U32 handle );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
Before calling the function, make sure application had called BmapiInitDiag() and BmapiSuspendDriverEx() to initialize diagnostic setup and suspend the NIC.
After the function returned, call BmapiResumeDriverEx() and BmapiUnInitDiag() to resume the NIC traffic and terminate diagnostic setup.
4401 doe not have CPU. CPU test simply return BMAPI_OK.

5.62 BmapiTestLEDsEx

The BmapiTestLEDsEx will flip the LEDs to indicate which logical NIC card is bound to physical NIC card for Broadcom made network adapters.

```
U32 BmapiTestLEDsEx(
  IN U32 handle,
  IN ULONG BlinkDurationSec );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

BlinkDurationSec

Specify how long to flip the LEDs (in seconds, up to 120 seconds).

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup. After the function returned, call BmapiUnInitDiag() to terminate diagnostic setup.

For 4401, applications must call BmapiSuspendDriverEx() prior to call the LED test.

5.63 BmapiSuspendDriverEx

BmapiSuspendDriverEx will suspend Broadcom miniport driver and bring down the link (as cable disconnected).

```
U32 BmapiSuspendDriverEx(
  IN U32 handle );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup. After the function returned, call BmapiUnInitDiag() to terminate diagnostic setup.

5.64 BmapiResumeDriverEx

BmapiResumeDriverEx will resume Broadcom miniport driver from suspend mode.

```
U32 BmapiResumeDriverEx(
  IN U32 handle );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.

Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup. After the function returned, call BmapiUnInitDiag() to terminate diagnostic setup.

5.65 BmapiGetFirmwareInfo

BmapiGetFirmwareInfo will get firmware information.

```
U32 BmapiGetFirmwareInfo(
  IN U32 handle,
  OUT BM_FW_INFO *pFWInfo );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pFWnfo

Pointer to the firmware information buffer.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401)

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information.
Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
Make sure set the ‘version’ in BM_FW_INFO to BMAPI_FW_INFO_VER before calling the function.
This API will go in diagnostics mode during execution if the application had not called BmapiInitDiag() prior to call this API.

5.66 BmapiTestASF

The BmapiTestASF will test various ASF related hardware components.

```
U32 BmapiTestASF(
  IN U32 handle );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetXtreme I (exclude 5700 and 5701)

Remarks:

BmapiTestASF() required driver to be suspended prior to the call.

5.67 BmapiWriteFirmware

BmapiWriteFirmware will write firmware information starting at specified offset with provided data buffer and lenngth to write to.

```
U32 BmapiWriteFirmware(
  IN U32 handle,
  IN U32 uOffset,
  IN U32 *pDataBuf,
  IN U32 uBufLen,
  IN U8 *pChk );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

uOffset

Starting offset to write data to.

pDataBuf

Data to write to firmware. Data are in 32-bit array.

uBufLen

Number of elements in data buffer array. For example, 2 means 2 32-bit data in the data buffer.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401), NetXtreme I, NetXtreme II

Remarks:

1. Applications need to call BmapiInitDiag() prior to call this and call BmapiUnInitDiag() after everything is done.
2. The newly programmed firmware will not be effective till:
   - The machine reboot.
   - The driver is disabled and re-enabled. Can be done manually or through BmapiEnableDevice().
   - Call BmapiSuspendDriverEx() followed by BmapiResumeDriverEx(). **This is the most preferrable method.**

5.68 BmapiReadFirmware

BmapiReadFirmware will read firmware information starting at specified offset with provided data buffer and length to read.

```
U32 BmapiReadFirmware(
  IN U32 handle,
  IN U32 uOffset,
  IN U32 *pDataBuf,
  IN U32 uBufLen,
  IN U8 *pChk );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

uOffset

Starting offset to read data from.

pDataBuf

Data buffer from firmware datra. Data are in 32-bit array.

uBufLen

Number of elements in data buffer array. For example, 2 means 2 32-bit data in the data buffer.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401), NetXtreme I, NetXtreme II

Remarks:

Applications need to call BmapiInitDiag() prior to call this and call BmapiUnInitDiag() after everything is done.

5.69 BmapiGetSystemASFTables

BmapiGetSystemASFTables will read ASF tables from system BIOS.

```
U32 BmapiGetSystemASFTables(
  IN BM_ASF_TABLE *pAsfTbls );
```

Parameters:

`pAsfTbls`

Pointer to ASF table structure that will be filled.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

Not applicable

Remarks:

Make sure set the ‘version’ in BM_ASF_TABLE to BMAPI_ASF_T_VERSION before calling the function.

5.70 BmapiGetBrcmVirNic

BmapiGetBrcmVirNic will return the information for BASP virtual adapters.

```
U32 BmapiGetBrcmVirNic(
  BM_VIR_NIC_INFO_EX *pNicInfoEx );
```

Parameters:

`handle`

Handle to the NIC information that should be retrieved.

`pNicInfoEx`

Pointer to the buffer for virtual NIC information.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

BASP created virtual adapters

Remarks:

Applications call BmapiGetAllTeam() to get all virtual adapters' basic information. If applications need more detail information, application will use the individual handle to call this function to retrieve information.

5.71 BmapiReadNicMem

BmapiReadNicMem will read network adapter's registers and memory.

```
U32 BmapiReadNicMem(
  IN U32 handle,
  IN U32 uType,
  IN U32 uOffset,
  IN U32 *pData,
  IN U8 *pChk );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

uType

Type of register or memory access.

uOffset

Offset to read from.

pData

Pointer to the return data.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401), NetXtreme I, NetXtreme II

Remarks:

1. The function always reads 4 bytes data at a time.
2. 'uType' could be BMAPI_INDIRECT_REG_READ, BMAPI_INDIRECT_MEM_READ or BMAPI_PHY_REG_READ.
3. Value pointed by 'pData' is valid only when the function returns BMAPI_OK.

5.72 BmapiWriteNicMem

BmapiWriteNicMem will write network adapter's registers and memory.

```
U32 BmapiWriteNicMem(
  IN U32 handle,
  IN U32 uType,
  IN U32 uOffset,
  IN U32 uData,
  IN U8 *pChk );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`uType`

Type of register or memory access.

`uOffset`

Offset to write to.

`uData`

Data to write to NIC.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401), NetXtreme I, NetXtreme II

Remarks:

1. The function always reads 4 bytes data at a time.
2. 'uType' could be BMAPI_INDIRECT_REG_READ, BMAPI_INDIRECT_MEM_READ or BMAPI_PHY_REG_READ.

5.73 BmapiGetIpAddrInfo

BmapiGetIpAddrInfo will read adapter's IP/NetMask/Gateway information.

```
U32 BmapiGetIpAddrInfo(
  IN U32 handle,
  IN U32 uType,
  OUT U8 *pBuf,
  IN OUT U32 *pLen );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

uType

Type of information to retrieve. It could be BMAPI_IPINFO_IP_LIST, BMAPI_IPINFO_SUBNETMASK_LIST or BMAPI_IPINFO_GATEWAY_LIST.

pBuf

Buffer to for retrieved information.

pLen

Pointer to a U32 number. On input, it is the length of buffer. On return, it is the required buffer length.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All NDIS adapters including NetLink, NetXtreme I, NetXtreme II L2 NDIS and BASP created virtual adapters

Remarks:

1. On input, uLen will be the length of buffer.
2. If 'pBuf' is NULL, 'uLen' will be the required length of buffer.
3. If the function failed due to buffer too short, 'uLen' will be the required length of buffer.
4. Returned data in 'pBuf' is in REG_MULTI_SZ format. (An array of null-terminated strings, terminated by two null characters.)

5.74 BmapiTestNetwork

BmapiTestNetwork will ping the destination IP.

```
U32 BmapiTestNetwork(
  IN U32 handle,
  IN U8 *pDestIP,
  IN U32 uRetry );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pDestIP

Destination IP string.

uRetry

How many times to retry.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All NDIS adapters including NetLink, NetXtreme I, NetXtreme II L2 NDIS and BASP created virtual adapters

Remarks:

The test will send ICMP packet to ping the destination. It will ping at least once and keep pinging till first response is received.
Application should NOT mix the test with other NIC diagnostic functions such as BmapiTestMIIRegistersEx().

5.75 BmapiGetPowerMode

BmapiGetPowerMode will get network adapter's current power saving mode from driver or EEPROM.

```
U32 BmapiGetPowerMode(
  IN U32 handle,
  IN U32 uWhere,
  IN U32 uPort,
  OUT U32 *pMode );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

uWhere

BMAPI_GET_POWER_MODE_DRIVER or
BMAPI_GET_POWER_MODE_EEPROM.

uPort

Reserved. MUST be 0.

pMode

Pointer to a U32 number. On return, it is the power mode setting of EEPROM or driver.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink and NetXtreme I

Remarks:

4401 adapter does not support setting in EEPROM (NVRAM).
If the application had called BmapiInitDiag() prior to call this API, BmapiInitDiag() must be called from the same thread that calls this API.

5.76 BmapiSetPowerMode

BmapiSetPowerMode will set network adapter's power saving mode from driver and/or EEPROM.

```
U32 BmapiSetPowerMode(
  IN U32 handle,
  IN U32 uWhere,
  IN U32 uPort,
  IN U32 uMode );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`uWhere`

BMAPI_SET_POWER_MODE_DRIVER,
BMAPI_SET_POWER_MODE_EEPROM or
BMAPI_SET_POWER_MODE_BOTH.

`uPort`

Reserved. MUST be 0.

`uMode`

The power mode setting to set to. It could be BMAPI_POWER_MODE_FULL or BMAPI_POWER_MODE_LOW.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink and NetXtreme I

Remarks:

4401 adapter does not support setting in EEPROM (NVRAM).
If the application had called BmapiInitDiag() prior to call this API, BmapiInitDiag() must be called from the same thread that calls this API.
When applications set the power mode to low, BMAPI will turn on power saving feature in firmware if it is disabled.

5.77 BmapiGetBRCMNicInfoEx

BmapiGetBRCMNicInfoEx will retrieve proprietary Broadcom network adapter information.

```
U32 BmapiGetBRCMNicInfoEx(
  IN U32 handle,
  OUT BM_BRCM_ADAPTER_INFO_EX *pBRCMNicInfoEx );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pBRCMNicInfoEx`

On input, applications pass a pointer to a BM_BRCM_ADAPTER_INFO_EX structure. On output, data will be filled out in the BM_BRCM_ADAPTER_INFO_EX structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink and NetXtreme I, NetXtreme II and its L2 NDIS

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.

Not all fields are applicable to 4401.

5.78 BmapiGetLastDiagPort

The API is obsolete.

5.79 BmapiWriteFirmwareInfo

BmapiWriteFirmwareInfo will update firmware information.

```
U32 BmapiWriteFirmwareInfo(
  IN U32 handle,
  IN BM_FW_INFO *pFWInfo,
  IN U32 uOption );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pFWInfo`

Pointer to BM_FW_INFO buffer will be copied to EEPROM.

`uOption`

Which block to update.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401 and selfboot), NetXtreme I

Remarks:

Currently, the function only supports BMAPI_WR_FW_MANUFAC option to update manufactory data block.

This API will go in diagnostics mode during execution if the application had not called BmapiInitDiag() prior to call this API.

5.80 BmapiGetTTBRCMNicInfo

The API is obsolete.

5.81 BmapiGetPHYStatus

BmapiGetPHYStatus will get current PHY status.

```
U32 BmapiGetPHYStatus(
  IN U32 handle,
  OUT U32 *pPhyStatus );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pPhyStatus`

Pointer to a U32 number. It will be filled current PHY status on function's return. The return value can be BMAPI_PHY_STATUS_OFF or BMAPI_PHY_STATUS_ON.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401 and 5700), NetXtreme I

Remarks:

If the application did not call BmapiInitDiag() before calling this API, BMAPI will enter diagnostics mode, query the information and leave diagnostics mode.

5.82 BmapiSetPHYStatus

BmapiSetPHYStatus will turn on or off PHY.

```
U32 BmapiSetPHYStatus(
  IN U32 handle,
  IN U32 uPhyStatus );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

uPhyStatus

The PHY mode setting to set to. Can be BMAPI_PHY_STATUS_OFF or BMAPI_PHY_STATUS_ON.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401 and 5700), NetXtreme I

Remarks:

If the application did not call BmapiInitDiag() before calling this API, BMAPI will enter diagnostics mode, query the information and leave diagnostics mode.

5.83 BmapiRetrieveLinkStatusEx

The BmapiRetrieveLinkStatusEx() will retrieve link status and link negotiation result for Broadcom made network adapters.

```
U32 BmapiRetrieveLinkStatusEx(
  IN U32 handle,
  OUT BM_LINK_STATUS_EX *pLinkStatusEx );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pLinkStatusEx

On input, applications pass a pointer to a BM_LINK_STATUS_EX structure. On output, data will be filled out in the BM_LINK_STATUS_EX structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtreme I, NetXtreme II and its L2 NDIS and third party NICs. All BASP created virtual adapters.

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

For all non-Broadcom made network adapters and BASP created virtual adapters, only "link_status", "line_speed_Kbps" and "driver_loaded" are available.

For 4401 network adapters, only "link_status", "duplex_mode", "link_speed" and fields for third party NICs are available.

5.84 BmapiTestLEDsEx2

The BmapiTestLEDsEx2() will flip the LEDs to indicate which logical NIC card is bound to physical NIC card for Broadcom made network adapters by blinking LEDs on the NIC.

```
U32 BmapiTestLEDsEx2(
  IN U32 handle,
  IN U32 BlinkDurationSec,
  IN U32 uFreq );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

BlinkDurationSec

Specify how long to flip the LEDs (in seconds, up to 120 seconds).

uFreq

Specify the frequency to blink the LEDs (from 1 to 10) in a second.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup. After the function returned, call BmapiUnInitDiag() to terminate diagnostic setup.

For 4401, applications must call BmapiSuspendDriverEx() prior to call the LED test.

5.85 BmapiGetNicStatistics64Ex

The function will return selected fields of statistics information from driver.

```
U32 BmapiGetNicStatistics64Ex(
  IN U32 handle,
  OUT BM_GENERAL_STATISTICS64 *pGenStatistics,
  OUT BM_ETHERNET_STATISTICS64 *pEthStatistics );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pGenStatistics

Pointer to a `BM_GENERAL_STATISTICS64` structure for general statistics information of a network adapter. The data will be filled out on return.

pEthStatistics

Pointer to an `BM_ETHERNET_STATISTICS64` structure for ethernet statistics information of a network adapter. The data will be filled out on return.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

All physical NICs including NetLink, NetXtreme I, NetXtreme II and its L2 NDIS and third party NICs. All BASP created virtual adapters.

Remarks:

Applications should allocate buffers for the statistics structures and pass pointers to those structures to the function. Application should be aware that some information might not be available. In case of NIC is disabled, the return code will be BMAPI_DRIVER_NOT_LOADED.

All fields in both BM_GENERAL_STATISTICS64 and BM_ETHERNET_STATISTICS64 structures MUST be initialized. Applications MUST choose which field will be read from driver by initializing the field with '0' or '-1' (all 'F'). '0' means the statistics data will be pulled from the driver. '-1' means the application does not need the field and the data will not be pulled from the driver. If a statistics data is not supported by the driver or not available for any reason, the corresponding field will be set to '-1'.

5.86 BmapiGetTeamSnapShot2

The function will return information for the teams.

```
U32 BmapiGetTeamSnapShot2(
  IN U32 teamId,
  OUT BM_TEAM_INFO2 *pTeam );
```

Parameters:

`teamId`

Handle to an adapter BM_ADAPTER_INFO structure.

`pTeam`

Pointer to BM_TEAM_INFO2 structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

N/A

Remarks:

Applications must call BmapiGetTeamListSnapShot() to learn available team information in driver. Applications, then, call BmapiGetTeamSnapShot2() to retrieve team information.

5.87 BmapiTeamFallbackPrimary

The function will force a SLB-Auto Fallback Disabled team to fallback to primary NICs.

```
U32 BmapiTeamFallbackPrimary(
    IN U32 teamId );
```

Parameters:

`teamId`

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

N/A

Remarks:

Applications must call BmapiGetTeamListSnapShot() to learn available team information in driver.

5.88 BmapiGetTeamDrvVersion

The function will return the version of BASP driver.

```
U32 BmapiGetTeamDrvVersion(
    OUT U32 pMajor,
    OUT U32 pMinor,
    OUT U32 pBuild );
```

Parameters:

pMajor

Pointer to major version number.

pMinor

Pointer to minor version number.

pBuild

Pointer to build version number.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

N/A

Remarks:

none

5.89 BmapiGetBRCMNicStatisticsEx

The BmapiGetBRCMNicStatisticsEx() will return Broadcom network adapters proprietary statistics data.

```
U32 BmapiGetBRCMNicStatisticsEx(
  IN U32 handle,
  IN/OUT BM_BRCM_STATISTICS_EX *pBrcmStatisticsEx );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pBrcmStatistics

Pointer to a BM_BRCM_STATISTICS_EX structure for Broadcom network adapter proprietary statistics information. The data will be filled out on return.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II

Remarks:

Applications should allocate buffers for the statistics structures and pass pointers to those structures to the function.

The 'version' field of BM_BRCM_STATISTICS_EX must be filled prior to call BmapiGetBRCMNicStatisticsEx(). The latest version is defined as BM_BRCM_STATISTICS_EX_VER in BMAPI.h.

5.90 BmapiGetBrcmNicParamList

The function will retrieve Broadcom network adapter's parameter list that are shown in the advance tab in NIC's propery page.

```
U32 BmapiGetBrcmNicParamList(
  IN U32 handle,
  IN U8 *pParamList,
 IN/OUT U32 *pLen );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pParamList

On input, applications pass a pointer to a buffer. On return the buffer will be filled with list of parameters' names in REG_MULTI_SZ format (an array of null-terminated strings, terminated by two null characters).

pLen

It is a pointer to a U32 number. On input, it is the length of buffer. On return, it is the required buffer length.

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II and its L2 NDIS

Remarks:

1. Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

2. If 'pBuf' is NULL, 'pLen' will return the required length of buffer.
3. If the function failed due to buffer too short, 'pLen' will be the required length of buffer.
4. Returned data in 'pParamList' is in REG_MULTI_SZ format. (an array of null-terminated strings, terminated by two null characters.)

5.91 BmapiGetBrcmNicParamInfo

The function will retrieve information of Broadcom network adapter's parameter.

```
U32 BmapiGetBrcmNicParamInfo(
  IN U32 handle,
  IN U8 *pParam,
  IN U8 *pCurVal,
  IN/OUT U32 *pCurValLen,
  IN U8 *pParamInfo,
  IN/OUT U32 *pParamInfoLen,
  IN U8 *pParamEnum,
  IN/OUT U32 *pParamEnumLen );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pParam

pointer to the name of the parameter

pCurVal

On input, applications pass a pointer to a buffer that will be filled with current value of the parameter.

pCurValLen

It is a pointer to a U32 number. On input, it is the length of buffer for 'pCurVal'. On return, it is the required buffer length for 'pCurVal'.

pParamInfo

On input, applications pass a pointer to a buffer. On return the buffer will be filled with list of the parameter's information in REG_MULTI_SZ format (an array of null-terminated strings, terminated by two null characters).

pParamInfoLen

It is a pointer to a U32 number. On input, it is the length of buffer for 'pParamInfo'. On return, it is the required buffer length for 'pParamInfo'.

pParamEnum

On input, applications pass a pointer to a buffer. On return the buffer will be filled with list of the parameter's enum information in REG_MULTI_SZ format (an array of null-terminated strings, terminated by two null characters).

pParamEnumLen

It is a pointer to a U32 number. On input, it is the length of buffer for 'pParamEnum'. On return, it is the required buffer length for 'pParamEnum'.

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II and its L2 NDIS

Remarks:

1. Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
2. If 'pCurVal' is NULL, 'pCurValLen' will return the required length of 'pCurVal'.
3. If 'pParamInfo' is NULL, 'pParamInfoLen' will return the required length of 'pParamInfo'.
4. If 'pParamEnum' is NULL, 'pParamEnumLen' will return the required length of 'pParamEnum'.
5. If the function failed due to buffer too short, 'pCurValLen', 'pParamInfoLen' and 'pParamEnumLen' will be the required length of their correspoding buffer.
6. Returned data in 'pParamInfo' and 'pParamEnum' are in REG_MULTI_SZ format (An array of null-terminated strings, terminated by two null characters.) with pairs of name and value information. For example, 'pParamInfo' of parameter 'Enable8021p' could be "ParamDesc\0802.1p QOS\0type\0enum\0\0" which means that 'ParamDesc' is set to value '802.1p QOS' (the string to display in GUI) and 'type' is set to value 'enum'. Another example, 'pParamEnum' of parameter 'Enable8021p' could be "0\0Disable\01\0Enable\0\0" which means that 'Enable8021p' could set to value 0 which means 'disable' and set to 1 which means 'enable'. One more example, 'pParamInfo' of parameter 'NetworkAddress' could be "Default\0\0type\0edit\0\0" which means that 'Default' is set to value '' (empty string) and 'type' is set to value 'edit'. Application MUST be able to handle the empty string case.
7. 'pParamEnum' information is available only when 'Type' in 'pParamInfo' is set to 'enum'.
8. The complete information including possible name and value of the information can be found in Microsoft's Windows 2000 DDK, 'Specifying Configuration Parameters for the Advanced Properties Page'.
9. If, currently, no 'pParam' is set in registry, '*pCurValLen' will be set to 0.
10. If pParamInfoLen is NULL, pParamInfo, pParamInfoLen, pParamEnum and pParamEnumLen will all NOT be filled with information. If pParamInfoLen is not NULL and pParamEnumLen is NULL, pParamEnum and pParamEnumLen will NOT be filled.

5.92 BmapiSetBrcmNicParam2

The function will retrieve information of Broadcom network adapter's parameter.

```
U32 BmapiSetBrcmNicParam2(
  IN U32 handle,
  IN U8 *pParam,
  IN pNewVal );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pParam

pointer to the name of the parameter

pNewVal

the new value to set for 'pParam'

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II and its L2 NDIS

Remarks:

1. Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
2. If pNewVal is NULL, the 'pParam' will be deleted from registry.

5.93 BmapiGetVbdEnumInfo

The function will retrieve VBD enumeration information.

```
U32 BmapiGetVbdEnumInfo(
  IN U32 handle,
  OUT BM_VBD_ENUM_INFO *pVbdInfo );
```

Parameters:

handle

Handle to a VBD adapter BM_ADAPTER_INFO structure.

pVbdInfo

On input, applications pass a pointer to BM_VBD_ENUM_INFO structure. On output, data will be filled out in the BM_VBD_ENUM_INFO structure.

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme II

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.

5.94 BmapiSetVbdEnumInfo

The function will set VBD enumeration information.

```
U32 BmapiSetVbdEnumInfo(
  IN U32 handle,
  IN BM_VBD_ENUM_INFO *pVbdInfo );
```

Parameters:

handle

Handle to a VBD adapter BM_ADAPTER_INFO structure.

pVbdInfo

On input, applications pass a pointer to BM_VBD_ENUM_INFO structure with the data that are supposed to set in VBD.

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme II

Remarks:

1. Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
2. If any change in device enumeration, application should reinitialize BMAPI after BmapiSetVbdEnumInfo() return BMAPI_OK in order to collect latest device information.
3. If the API returns BMAPI_REBOOT_REQUIRED, the system needs a reboot for the change to take effect.

5.95 BmapiGetLicenseKey

The function will return specific license information for the NIC.

```
U32 BmapiGetLicenseKey(
  IN U32 handle,
  IN U32 key_index,
  OUT BMAPI_LICENSE_INFO *pLicense );
```

Parameters:

handle

Handle to a VBD adapter BM_ADAPTER_INFO structure.

key_index

Index of the key. Current defined keys include BMAPI_LICENSE_KEY_IDX_MANUFAC and BMAPI_LICENSE_KEY_IDX_UPGRADE.

pLicense

Pointer to a buffer (BMAPI_LICENSE_INFO) that will filled with license key information.

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.

5.96 BmapiSetLicenseKey

The function will set specific license information for the NIC.

```
U32 BmapiSetLicenseKey(
  IN U32 handle,
  IN U32 key_index,
  IN U8 *key_buf,
  IN U32 key_len );
```

Parameters:

handle

Handle to a VBD adapter BM_ADAPTER_INFO structure.

key_index

Index of the key. Currently support only BMAPI_LICENSE_KEY_IDX_UPGRADE.

key_buf

Pointer to a buffer that contains license key information.

key_len

length of license key buffer (key_buf).

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme II 5706 family

Remarks:

1. Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
2. If the API returns BMAPI_REBOOT_REQUIRED, the system needs a reboot for the change to take effect.

5.97 BmapiGetResourceConfig

The function will get resource configuration for the NIC.

```
U32 BmapiGetResourceConfig(
  IN U32 handle,
  OUT BM_RES_CFG *pResCfg );
```

Parameters:

handle

Handle to a VBD adapter BM_ADAPTER_INFO structure.

pResCfg

pointer to BM_RES_CFG buffer that will be filled on return.

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.98 BmapiSetResourceConfig

The function will set resource configuration for the NIC.

```
U32 BmapiSetResourceConfig(
  IN U32 handle,
  IN BM_RES_CFG *pResCfg );
```

Parameters:

`handle`

Handle to a VBD adapter BM_ADAPTER_INFO structure.

`pResCfg`

pointer to BM_RES_CFG buffer that with data to set

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme II 5706 family

Remarks:

1. Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.
2. Only 'toe_reserved_con', 'rdma_reserved_con', 'iscsi_reserved_con' and 'iser_reserved_con' will be used to set the new configuration.
3. The reserved connections will be set ONLY when the stack of the technology is to be enumerated by VBD. In other words, if a technology is either not licensed or not enumerated by VBD, the reserved connedtions will not be set. For example, if NDIS stack is not enabled, reserved TOE connections will be set to 0.
4. If the API returns BMAPI_REBOOT_REQUIRED, the system needs a reboot for the change to take effect.
5. Meaning of some other possible return codes:
    - BMAPI_CON_EXCEED_HW_MAX: one or more reserved connections exceed hardware limitation
    - BMAPI_CON_EXCEED_LIC_MAX: one or more reserved connections exceed lincensed connections
    - BMAPI_RES_EXCEED_HW_MAX: total reserved resources exceed limitation

5.99 BmapiGetTeamIDList

The function will return to caller for IDs of teams configured.

```
U32 BmapiGetTeamIDList(
  IN U32 uNumOfTeam,
  IN OUT U32 *pIDList );
```

Parameters:

uNumOfTeam

Pointer to number of U32 allocated in pIDList array.

pIDList

Pointer to buffers for IDs of team configured. IDs will be filled on return.

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

N/A

Remarks:

Applications must call BmapiGetNumTeam() to understand how many teams are in driver. Applications, then, allocate enough buffers to fill all ID and call BmapiGetTeamIDList() to Team IDs. If uNumOfTeam is not big enough, the function will return error code. After IDs were retrieved, applications can call BmapiGetTeamInfo() to get information of a team.

5.100 BmapiGetTeamInfo

The function will return a team information specified by parameter 'team_id'.

```
U32 BmapiGetTeamInfo(
  IN U32 team_id,
  IN OUT BM_TEAM_INFO2 *pTeamInfo2 );
```

Parameters:

team_id

ID of the team to get information for

pTeamInfo2

Pointer to the buffer for the team information which will be filled on return

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

N/A

Remarks:

Applications must call BmapiGetNumTeam() to understand how many teams are in driver. Applications, then, allocate enough buffers to fill all ID and call BmapiGetTeamIDList() to Team IDs. If uNumOfTeam is not big enough, the function will return error code. After IDs were retrieved, applications can call BmapiGetTeamInfo() to get information of a team.

5.101 BmapiApplyLBFOCfgEx

The function will apply configuration changes.

```
U32 BmapiApplyLBFOCfgEx(
  IN BM_TEAM_INFO2 **ppTeamList,
  IN U32 uNumOfTeam,
  IN OUT U32 *pReboot );
```

Parameters:

`ppTeamList`

Pointer to the pointer array of BM_TEAM_INFO2 for all teams

`uNumOfTeam`

Number of pointers in 'ppTeamList' array

`pReboot`

Pointer to a boolean flag to indicate system need to reboot or not

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

N/A

Remarks:

The function may trigger binding if it is necessary. If return value pointed by 'pReboot' is non-zero, reboot is required.

5.102 BmapiGetTeamStatisticsSnapShotEx

The function will return statistics information for a team from driver.

```
U32 BmapiGetTeamStatisticsSnapShotEx(
  IN U32 teamId,
  IN OUT BM_TEAM_STATISTICS_EX *pTeamStat );
```

Parameters:

`teamId`

Team Id of the target team

`pTeamStat`

Pointer to BM_TEAM_STATISTICS_EX structure

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

N/A

Remarks:

Applications must call BmapiGetTeamListSnapShot() to learn available team information in driver. Applications, then, call BmapiGetTeamStatisticsSnapShot() to retrieve statistics information of a team.

5.103 BmapiGet5706FwInfo

The function will get firmware information for 5706 family NICs.

```
U32 BmapiGet5706FwInfo(
  IN U32 handle,
  IN OUT BM_FW_INFO_5706 *pFwInfo );
```

Parameters:

handle

Handle to the adapter

pFwInfo

Pointer to BM_FW_INFO_5706 buffer that will be filled on return

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.104 BmapiTestLEDsAsyncStart

The function will blink LEDs of the NIC at the provided frequency in asynchronous style.

```
U32 BmapiTestLEDsAsyncStart(
  IN U32 handle,
  IN U32 uFreq );
```

Parameters:

handle

Handle to the adapter

uFreq

The frequency of the LEDs to blink. (1 to 10).

Return Value:

Return BMAPI_OK if the LED blinking thread is successfully started, otherwise a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

1. Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
2. The LEDs will be kept blinking till
   - BmapiTestLEDsAsyncStop() is called
   - BmapiUnInitDiag() is called
   - BMAPI is uninitialized and instance count down to zero
   - the process exits out properly

5.105 BmapiTestLEDsAsyncStop

The function will stop async. LEDs blinking operation.

```
U32 BmapiTestLEDsAsyncStop(
  IN U32 handle,
  IN OUT U32 *pResult );
```

Parameters:

handle

Handle to the adapter

pResult

The return code of the LED blinking operation

Return Value:

Return BMAPI_OK if the LED blinking stop properly and result code is retrieved, otherwise a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II 5706 family

Remarks:

Applications must call BmapiGetAllPhyNic(), etc. to get adapter information. Applications will find the ‘handle’ in BM_ADAPTER_INFO and use it to call the function.

5.106 BmapiGetASFMailboxCount

BmapiGetASFMailboxCount will return the number of supported mailboxes.

```
U32 BmapiGetASFMailboxCount(
  IN U32 handle,
  OUT U32 *pCount,
  OUT U32 *pTotalMbxLen );
```

Parameters:

`handle`

handle to the adapter

`pCount`

number of supported mailboxes

`pTotalMbxLen`

total number of bytes for all mailboxes in NVRAM

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme I ASF cabpable NICs

Remarks:

1. Application must call BmapiInitDiag() before calling this API.
2. 'pTotalMbxLen' is optional. If it is NULL, the data will not be returned.

5.107 BmapiGetASFMailboxStatus

BmapiGetASFMailboxStatus will return the status of the mailbox.

```
U32 BmapiGetASFMailboxStatus(
  IN U32 handle,
  IN U32 uMboxNum,
  OUT BM_ASF_MBOX_STATUS *pMboxStatus );
```

Parameters:

`handle`

handle to the adapter

`uMboxNum`

number the mailbox to read

pMboxStatus

buffer to the mailbox status

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme I ASF cabpable NICs

Remarks:

1. Application must call BmapiInitDiag() before calling this API.
2. Mailbox number is '0' based.

5.108 BmapiSetASFMailboxStatus

BmapiSetASFMailboxStatus will set the status of the mailbox.

```
U32 BmapiSetASFMailboxStatus(
  IN U32 handle,
  IN U32 uMboxNum,
  IN BM_ASF_MBOX_STATUS *pMboxStatus );
```

Parameters:

handle

handle to the adapter

uMboxNum

number the mailbox to set data to

pMboxStatus

buffer to the mailbox status

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme I ASF cabpable NICs

Remarks:

1. Application must call BmapiInitDiag() before calling this API.
2. Mailbox number is '0' based.
3. 'length' field will be ignored in BmapiSetASFMailboxStatus().

5.109 BmapiGetASFMailboxContents

BmapiGetASFMailboxContents will get contents of the mailbox.

```
U32 BmapiGetASFMailboxContents(
  IN U32 handle,
  IN U32 uMboxNum,
  IN U32 uOffset,
  IN OUT U32 *pLength,
  OUT U8 *pBuf );
```

Parameters:

`handle`

handle to the adapter

`uMboxNum`

number the mailbox to read data from

`uOffset`

offset in the content to read the data

`pLength`

pointer to the length of data to read (bytes)

`pBuf`

pointer to the buffer for the data being read

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme I ASF cabpable NICs

Remarks:

1. Application must call BmapiInitDiag() before calling this API.
2. Mailbox number is '0' based.
3. If 'uOffset' + '*pLength' is longer than 'fill_level' of the mailbox, '*pLength' will set to 'fill_level' - 'uOffset'.

5.110 BmapiSetASFMailboxContents

BmapiSetASFMailboxContents will set contents of the mailbox.

```
U32 BmapiSetASFMailboxContents(
  IN U32 handle,
  IN U32 uMboxNum,
  IN U32 uOffset,
  IN U32 uLength,
  IN U8 *pBuf );
```

Parameters:

handle

handle to the adapter

uMboxNum

number the mailbox to set data to

uOffset

offset in the content to write the data

uLength

length of data to write (bytes)

pBuf

pointer to the buffer for the data to be written to the mailbox

Return Value:

Return BMAPI_OK if successful, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme I ASF cabpable NICs

Remarks:

1. Application must call BmapiInitDiag() before calling this API.
2. Mailbox number is '0' based.
3. The 'fill_level' in status block of the mailbox will NOT be updated.
4. 'uOffset' + 'uLength' can not be greater than existing 'length' in the status of the mailbox.

5.111 BmapiTestASFMailboxes

BmapiTestASFMailboxes will test the integrity of ASF mailbox.

```
U32 BmapiTestASFMailboxes(
  IN U32 handle );
```

Parameters:

handle

handle to the adapter

Return Value:

Return BMAPI_OK if the ASF mailbox appears to be ok, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme I ASF cabpable NICs

Remarks:

1. Application must call BmapiInitDiag() before calling this API.
2. Mailbox number is '0' based.

5.112 BmapiTestCable

BmapiTestCable will perform cable diagnostics for the nic.

```
U32 BmapiTestCable(
  IN U32 handle,
  OUT BMAPI_CAB_DIAG *pCabDiag );
```

Parameters:

`handle`

handle to the adapter

`pCabDiag`

pointer to cable diagnostics result

Return Value:

Return BMAPI_OK if the ASF mailbox appears to be ok, otherwise a nonzero error code.

Supported Network Adapters:

5754 family, 5755 family, NetXtreme II (copper based)

Remarks:

1. Before calling the function, make sure application had called BmapiInitDiag() to initialize diagnostic setup and called BmapiSuspendDriverEx() to stop traffic on the NIC. After the function returned, call BmapiResumeDriverEx() to resume traffic and call BmapiUnInitDiag() to terminate diagnostic setup.
2. If BMAPI_OK is returned from the API, the application should check 'status' for each channel to determine the result for each channel.
3. The definition of 'status' is defined in BMAPI.h and they look like 'BMAPI_CABDIAG_STATUS_XXXXX'.
4. The 'distanceCm' will contain the length of the channel only when the 'status' of the channel indicates that length is available.

5.113 BmapiGetOffloadStackInfo

BmapiGetOffloadStackInfo() will return oflload stack information from driver.

```
U32 BmapiGetOffloadStackInfo(
  IN U32 handle,
  IN U32 uStackIdx,
  OUT BMAPI_OFLD_STACK_INFO *pStackInfo );
```

Parameters:

`handle`

handle to the adapter

`uStackIdx`

which stack to query

`pStackInfo`

pointer to a BMAPI_OFLD_STACK_INFO structure for the returned stack information.

Return Value:

Return BMAPI_OK if the ASF mailbox appears to be ok, otherwise a nonzero error code.

Supported Network Adapters:

NetXtreme II

Remarks:

Applications should allocate buffers for the statistics structures and pass pointers to those structures to the function.

5.114 BmapiGetISCSIConfig

BmapiGetISCSIConfig() will retrieve iSCSI configuration..

```
U32 BmapiGetISCSIConfig(
  IN U32 handle,
  OUT BMAPI_ISCSI_CONFIG *pConfig );
```

Parameters:

`handle`

handle to the VBD adapter

`pConfig`

On input, applications pass a pointer to BMAPI_ISCSI_CONFIG structure. On return, data will be filled out in the BMAPI_ISCSI_CONFIG structure.

Return Value:

Return BMAPI_OK if iSCSI configuration is retrieved successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme II

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.115 BmapiSetISCSIConfig

BmapiSetISCSIConfig() will set iSCSI configuration.

```
U32 BmapiSetISCSIConfig(
  IN U32 handle,
  IN BMAPI_ISCSI_CONFIG *pConfig );
```

Parameters:

`handle`

handle to the VBD adapter

`pConfig`

Applications pass a pointer to BMAPI_ISCSI_CONFIG structure with the data that are supposed to set.

Return Value:

Return BMAPI_OK if iSCSI configuration is updated successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme II

Remarks:

- Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
- If any change in iSCSI configuration, application should re-start iSCSI stack to make new configuration taking effect.

5.116 BmapiGet57710FwInfo

BmapiGet57710FwInfo() will get firmware information for 57710 family NICs.

```
U32 BmapiGet57710FwInfo(
  IN U32 handle,
  OUT BM_FW_INFO_57710 *pFwInfo );
```

Parameters:

`handle`

handle to the VBD adapter

`pFwInfo`

Applications pass a pointer to BM_FW_INFO_57710 buffer that will be filled on return.

Return Value:

Return BMAPI_OK if firmware information is retrieved successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme II 57710 family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.117 BmapiGetMgmtProcessors

BmapiGetMgmtProcessors() will return the type of management processors on the controllers.

```
U32 BmapiGetMgmtProcessors(
  IN U32 handle,
  OUT U32 *pProc );
```

Parameters:

handle

handle to a NIC port

pProc

pointer to a U32 number. The number is a bit definition to various types of management processors in our controllers.

Return Value:

Return BMAPI_OK if information was retrieved successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme I family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.118 BmapiGetMgmtEnableState

BmapiGetMgmtEnableState() will return whether management FW is enabled or not.

```
U32 BmapiGetMgmtEnableState(
  IN U32 handle,
  OUT U32 *pEnable );
```

Parameters:

`handle`

handle to a NIC port

`pEnable`

pointer to a U32 number. Set to '0' if management FW is disabled in NVRAM. Non-zero value if it is enabled in NVRAM.

Return Value:

Return BMAPI_OK if information was retrieved successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme I family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.119 BmapiSetMgmtEnableState

BmapiSetMgmtEnableState() will set management FW to enable or disable.

```
U32 BmapiSetMgmtEnableState(
  IN U32 handle,
  IN U32 bEnable );
```

Parameters:

`handle`

handle to a NIC port

`bEnable`

'0' to disable management FW and non-zero to enabled.

Return Value:

Return BMAPI_OK if configuration was set successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme I family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.120 BmapiAssertMgmtEvent

BmapiAssertMgmtEvent() will assert an APE event.

```
U32 BmapiAssertMgmtEvent(
  IN U32 handle,
  IN U8 uEventID,
  IN U8 uEventData,
  IN U8 Reserved,
  IN void *pMsg,
  IN U32 uMsgLen );
```

Parameters:

handle

handle to a NIC port

uEventID

Event ID

uEventData

Event data

Reserved

reserved

**pMsg*

Pointer to the message for the event

uMsgLen

length of the message in ‘pMsg’

Return Value:

Return BMAPI_OK if event was asserted successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.121 BmapiGetMgmtOTPKeys

BmapiGetMgmtOTPKeys() will read OTP keys.

```
U32 BmapiGetMgmtOTPKeys(
  IN U32 handle,
  OUT U32 *pKey1,
  OUT U32 *pKey2 );
```

Parameters:

`handle`

Handle to a NIC port

`pKey1`

Pointer to the first part of the key on return

`pKey2`

Pointer to the second part of the key on return

Return Value:

Return BMAPI_OK if keys were read retrieved successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.122 BmapiGetMgmtDataLength

BmapiGetMgmtDataLength() will get the length of APE data.

```
U32 BmapiGetMgmtDataLength(
  IN U32 handle,
  OUT U32 *pLength );
```

Parameters:

`handle`

Handle to a NIC port

`pLength`

Pointer to the APE data length (filled on return)

Return Value:

Return BMAPI_OK if length was read successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.123 BmapiGetMgmtData

BmapiGetMgmtData() will get APE data.

```
U32 BmapiGetMgmtData(
  IN U32 handle,
  IN U32 uStart,
  OUT void *pBuf,
  IN U32 uBufLen );
```

Parameters:

`handle`

Handle to a NIC port

`uStart`

Starting offset to read APE data (from beginning of APE data)

`pBuf`

Pointer to the APE data (filled on return)

`uBufLen`

Length of buffer (‘pBuf’)

Return Value:

Return BMAPI_OK if APE data is read successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.124 BmapiSetMgmtData

BmapiSetMgmtData() will set APE data.

```
U32 BmapiSetMgmtData(
  IN U32 handle,
  IN U32 uStart,
  IN void *pBuf,
  IN U32 uBufLen );
```

Parameters:

`handle`

Handle to a NIC port

`uStart`

Starting offset to write APE data (from beginning of APE data)

`pBuf`

Pointer to the APE data that will write to the NIC

`uBufLen`

Length of buffer (‘pBuf’) to write to NIC

Return Value:

Return BMAPI_OK if APE data is written successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.125 BmapiGetMgmtConfigLength

BmapiGetMgmtConfigLength() will get the length of management configuration data.

```
U32 BmapiGetMgmtConfigLength(
  IN U32 handle,
  OUT U32 *pLength );
```

Parameters:

`handle`

Handle to a NIC port

`pLength`

Pointer to the management configuration data length (filled on return)

Return Value:

Return BMAPI_OK if the length is retrieved successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme I family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.126 BmapiGetMgmtConfig

BmapiGetMgmtConfig() will read management configuration data.

```
U32 BmapiGetMgmtConfig(
  IN U32 handle,
  OUT void *pBuf,
  IN U32 uBufLen );
```

Parameters:

`handle`

Handle to a NIC port

`pBuf`

Pointer to the management configuration data (filled on return)

`uBufLen`

Length of the buffer for management configuration data (‘pBuf’)

Return Value:

Return BMAPI_OK if the management configuration data is read successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme I family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.127 BmapiSetMgmtConfig

BmapiSetMgmtConfig() will write management configuration data.

```
U32 BmapiSetMgmtConfig(
  IN U32 handle,
  IN void *pBuf,
  IN U32 uBufLen );
```

Parameters:

handle

Handle to a NIC port

pBuf

Pointer to the management configuration data that will write to the NIC

uBufLen

Length of buffer for management configuration data to write

Return Value:

Return BMAPI_OK if the management configuration data is written to the NIC successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme I family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.128 BmapiGetMgmtSharedMem

BmapiGetMgmtSharedMem() will read APE shared memory.

```
U32 BmapiGetMgmtSharedMem(
  IN U32 handle,
  IN U32 uStart,
  IN void *pBuf,
  IN U32 uBufLen );
```

Parameters:

handle

Handle to a NIC port

uStart

Starting offset to read APE shared memory

pBuf

Pointer to the APE shared memory data (filled on return)

uBufLen

Length of buffer (‘pBuf’) in bytes, must be multiple of DWORD

Return Value:

Return BMAPI_OK if APE shared memory is read successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.129 BmapiGetISCSIConfig2

BmapiGetISCSIConfig2() will retrieve iSCSI configuration.

```
U32 BmapiGetISCSIConfig2(
  IN U32 handle,
  OUT BMAPI_ISCSI_CONFIG *pConfig,
  OUT BM_ISCIS_IPV6_ADDR *pStaticIPv6,
  IN/OUT U32 *pStaticIPv6Len,
  OUT BM_ISCIS_IPV6_ADDR *pGatewayIPv6,
  IN/OUT U32 *pGatewayIPv6Len );
```

Parameters:

handle

Handle to a VBD

pConfig

On input, applications pass a pointer to BMAPI_ISCSI_CONFIG structure. On return, data will be filled out in the BMAPI_ISCSI_CONFIG structure.

pStaticIPv6

On input, applications pass a pointer to BM_ISCIS_IPV6_ADDR structure(s). On return, data will be filled out. If *pStaticIPv6Len is 0, 'pStaticIPv6' will be ignored.

pStaticIPv6Len

On input, applications pass a pointer to the buffer length of 'pStaticIPv6'. On return, the actual length of 'pStaticIPv6' will be retyrned. If the value of the length is 0 at input, the required length will be returned and 'pStaticIPv6' will be ignored.

pGatewayIPv6

On input, applications pass a pointer to BM_ISCIS_IPV6_ADDR structure(s). On return, data will be filled out. If *pGatewayIPv6Len is 0, 'pGatewayIPv6' will be ignored.

`pGatewayIPv6Len`

On input, applications pass a pointer to the buffer length of 'pGatewayIPv6'. On return, the actual length of 'pStaticIPv6' will be retyrned. If the value of the length is 0 at input, the required length will be returned and 'pGatewayIPv6' will be ignored.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetXtreme II family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.130 BmapiSetISCSIConfig2

BmapiSetISCSIConfig2() will set iSCSI configuration.

```
U32 BmapiSetISCSIConfig2(
  IN U32 handle,
  OUT BMAPI_ISCSI_CONFIG *pConfig,
  OUT BM_ISCIS_IPV6_ADDR *pStaticIPv6,
  IN U32 uStaticIPv6Len,
  OUT BM_ISCIS_IPV6_ADDR *pGatewayIPv6,
  IN U32 uGatewayIPv6Len );
```

Parameters:

`handle`

Handle to a VBD

`pConfig`

a pointer to BMAPI_ISCSI_CONFIG structure with the data that will be set.

`pStaticIPv6`

a pointer to BM_ISCIS_IPV6_ADDR structure(s) with the data that will be set.

`uStaticIPv6Len`

buffer length of 'pStaticIPv6'.

`pGatewayIPv6`

a pointer to BM_ISCIS_IPV6_ADDR structure(s) with the data that will be set.

`uGatewayIPv6Len`

buffer length of 'pGatewayIPv6'.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetXtreme II family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

If any change in iSCSI configuration, application should re-start iSCSI stack to make new configuration taking effect.

5.131 BmapiWritePhyFirmware

BmapiWritePhyFirmware() writes PHY firmware to PHY devices.

```
U32 BmapiWritePhyFirmware(
  IN U32 handle,
  IN U32 *pDataBuf,
  IN U32 uBufLen,
  IN U8 *pChk );
```

Parameters:

handle

Handle to a VBD

pDataBuf

pointer to PHY firmware.

uBufLen

length of PHY firmware in 'pDataBuf' (in byte).

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

57710 family

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.132 BmapiCreateMgmtData

BmapiCreateMgmtData() will create APE_DATA section in NVRAM. If the APE_DATA exists, it will be resized to 'uLength'. The content of APE_DATA should be zeroed-out.

```
U32 BmapiCreateMgmtData(
  IN U32 handle,
  IN U32 uLength );
```

Parameters:

`handle`

Handle to a NIC port

`uLength`

length of APE_DATA

Return Value:

Return BMAPI_OK if APE_DATA created, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.133 BmapiGetMgmtWebDataLength

BmapiGetMgmtWebDataLength()will get the length of APE WEB data (if it exists).

```
U32 BmapiGetMgmtWebDataLength(
  IN U32 handle,
  OUT U32 *pLength );
```

Parameters:

`handle`

Handle to a NIC port

`pLength`

pointer to the APE WEB data length (filled on return)

Return Value:

Return BMAPI_OK if length was read successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.134 BmapiGetMgmtWebData

BmapiGetMgmtWebData() will get APE WEB data.

```
U32 BmapiGetMgmtWebData(
  IN U32 handle,
  IN U32 uStart,
  OUT void *pBuf,
  IN U32 uBufLen );
```

Parameters:

handle

Handle to a NIC port

uStart

Starting offset to read APE WEB data (from beginning of APE WEB data)

pBuf

Pointer to the APE WEB data (filled on return)

uBufLen

Length of buffer (‘pBuf’)

Return Value:

Return BMAPI_OK if APE WEB data is read successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.135 BmapiSetMgmtWebData

BmapiSetMgmtWebData() will set APE WEB data.

```
U32 BmapiSetMgmtWebData(
  IN U32 handle,
  IN U32 uStart,
  IN void *pBuf,
  IN U32 uBufLen );
```

Parameters:

handle

Handle to a NIC port

uStart

Starting offset to write APE WEB data (from beginning of APE WEB data)

pBuf

Pointer to the APE WEB data that will write to the NIC

uBufLen

Length of buffer (‘pBuf’) to write to NIC

Return Value:

Return BMAPI_OK if APE WEB data is written successfully, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

1. Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
2. ‘uBufLen’ must be in multiple of 4 bytes.

5.136 BmapiCreateMgmtWebData

BmapiCreateMgmtWebData() will create APE_WEB_DATA section in NVRAM. If the APE_WEB_DATA exists, it will be resized to 'uLength'. The content of APE_WEB_DATA should be zeroed-out.

```
U32 BmapiCreateMgmtWebData(
  IN U32 handle,
  IN U32 uLength );
```

Parameters:

handle

Handle to a NIC port

uLength

length of APE_WEB_DATA

Return Value:

Return BMAPI_OK if APE_WEB_DATA created, otherwise a nonzero error code will be returned.

Supported Network Adapters:

DASH supported chips

Remarks:

Applications must call BmapiGetAllPhyNic() or BmapiGetPhyNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.

5.137 BmapiRetrieveMultiLinkStatus

BmapiRetrieveMultiLinkStatus() will return multiple link information.

```
U32 BmapiRetrieveMultiLinkStatus(
  IN/OUT BM_LINK_STATUS_EX **ppLinkStatusEx,
  IN U32 uNumOfStruct );
```

Parameters:

`ppLinkStatusEx`

Pointer to the pointer array of BM_LINK_STATUS_EX for all link status structures that need to be filled.

`uNumOfStruct`

Number of BM_LINK_STATUS_EX structures in 'ppLinkStatusEx' array.

Return Value:

Return BMAPI_OK if all information were collected, otherwise a nonzero error code will be returned.

Supported Network Adapters:

All

Remarks:

1. Applications must call BmapiGetAllPhyNic() or BmapiGetAllUnassignedNic() to get adapter information. Applications will find the 'handle' in BM_ADAPTER_INFO and use it to call the function.
2. On input, 'version' and 'handle' fields must be provided.

5.138 BmapiGetISCSIRuntimeIPCount

BmapiGetISCSIRuntimeIPCount() will retrieve number of IPV4/IPV6 addresses configured for one or more iSCSI ports.

```
U32 BmapiGetISCSIRuntimeIPCount(
  IN/OUT BM_ISCSI_IP_COUNT *pIpCount,
  IN U32 uCount );
```

Parameters:

pIpCount

On input, applications pass a pointer to empty BM_ISCSI_IP_COUNT array. On return, data will be filled out.

uCount

number of BM_ISCSI_IP_COUNT in 'pIpCount'.

Return Value:

Return BMAPI_OK if information were collected, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme II

Remarks:

The caller of the function must call CoInitializeEx() to initialize COM and call CoInitializeSecurity() to initialize security values for the process prior to calling the function.

5.139 BmapiGetISCSIRuntimeIP

BmapiGetISCSIRuntimeIP() will retrieve iSCSI runtime IP information.

```
U32 BmapiGetISCSIRuntimeIP(
  IN/OUT BM_ISCSI_IPV4_RT **ppIPv4,
  IN U32 uIPv4Len,
  IN/OUT BM_ISCSI_IPV6_RT **ppIPv6,
  IN U32 uIPv6Len );
```

Parameters:

ppIPv4

On input, applications pass a pointer to array of BM_ISCSI_IPV4_RT pointers. On return, data will be filled out.

uIPv4Len

Number of pointers to BM_ISCSI_IPV4_RT passed down in 'ppIPv4'.

ppIPv6

On input, applications pass a pointer to array of BM_ISCSI_IPV6_RT pointers. On return, data will be filled out.

uIPv6Len

Number of pointers to BM_ISCSI_IPV4_RT passed down 'ppIPv6'.

Return Value:

Return BMAPI_OK if information were collected, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme II

Remarks:

The caller of the function must call CoInitializeEx() to initialize COM and call CoInitializeSecurity() to initialize security values for the process prior to calling the function.

5.140 BmapiGetISCSIRuntimeStatistics

BmapiGetISCSIRuntimeStatistics() will retrieve iSCSI runtime statistics information.

```
U32 BmapiGetISCSIRuntimeStatistics(
  IN/OUT BM_ISCSI_STATS **ppStats,
  IN U32 U32 uStatsLen );
```

Parameters:

ppStats

On input, applications pass a pointer to array of BM_ISCSI_STATS pointers. On return, data will be filled out.

uStatsLen

Number of pointers to BM_ISCSI_STATS passed down in 'ppStats'

Return Value:

Return BMAPI_OK if information were collected, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme II

Remarks:

The caller of the function must call CoInitializeEx() to initialize COM and call CoInitializeSecurity() to initialize security values for the process prior to calling the function.

5.141 BmapiGetISCSISessionStatistics

BmapiGetISCSISessionStatistics() will retrieve iSCSI session statistics information.

```
U32 BmapiGetISCSISessionStatistics(
  IN/OUT BM_ISCSI_SESSION_STATS **ppSessionStats,
  IN U32 U32 uStatsLen );
```

Parameters:

`ppSessionStats`

On input, applications pass a pointer to array of BM_ISCSI_SESSION_STATS pointers. On return, data will be filled out.

`uStatsLen`

Number of pointers to BM_ISCSI_SESSION_STATS passed down in 'ppSessionStats'.

Return Value:

Return BMAPI_OK if information were collected, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme II

Remarks:

The caller of the function must call CoInitializeEx() to initialize COM and call CoInitializeSecurity() to initialize security values for the process prior to calling the function.

5.142 BmapiResetNdisStatistics

BmapiResetNdisStatistics() will reset NDIS statistics from driver.

```
U32 BmapiResetNdisStatistics(
  IN U32 handle );
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

Return Value:

Return BMAPI_OK if reset is done, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetXtreme I and NDIS nodes from NetXtreme II

Remarks:

1. The handle must belong to an NDIS device. For NX2, it must belong to the NDIS child node.
2. The API does not support WinPE environment.

5.143 BmapiWriteFirmware2

BmapiWriteFirmware2() will write firmware information starting at specified offset with provided data and len to write to.

```
U32 BmapiWriteFirmware2(
  IN U32 handle,
  IN U32 uOffset,
  IN U32 *pDataBuf,
  IN U32 uBufLen,
  IN U8 *pChk,
  IN U32 uTarget );
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

uOffset

Starting offset to write data to.

pDataBuf

Data to write to firmware. Data are in 32-bit array.

uBufLen

Number of elements in data buffer array. For example, 2 means 2 32-bit data in the data buffer.

uTarget

BMAPI_NVRAM_TARGET_ACTIVE,
BMAPI_NVRAM_TARGET_NVRAM or
BMAPI_NVRAM_TARGET_OTP.

Return Value:

Return BMAPI_OK if reset is done, otherwise a nonzero error code will be returned.

Supported Network Adapters:

NetLink (exclude 4401), NetXtreme I, NetXtreme II

Remarks:

1. Applications need to call BmapiInitDiag() prior to call this and call BmapiUnInitDiag() after everything is done.

2. The newly programmed firmware will not be effective till:
    - The machine reboot.
    - The driver is disabled and re-enabled. Can be done manually or through BmapiEnableDevice().
    - Call BmapiSuspendDriverEx() followed by BmapiResumeDriverEx(). **This is the most preferrable method.**
3. ‘uTarget’ is applicable only to NetLink and NetXtreme.
4. BMAPI_NVRAM_TARGET_OTP is applicable only to some NetXtreme devices only.

5.144 BmapiReadFirmware2

BmapiReadFirmware2() will read firmware information starting at specified offset with provided data buffer and length to read.

```
U32 BmapiReadFirmware2(
  IN U32 handle,
  IN U32 uOffset,
  IN U32 *pDataBuf,
  IN U32 uBufLen,
  IN U8 *pChk,
  IN U32 uTarget )
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

uOffset

Starting offset to read data from.

pDataBuf

Data buffer from firmware datra. Data are in 32-bit array.

uBufLen

Number of elements in data buffer array. For example, 2 means 2 32-bit data in the data buffer.

uTarget

BMAPI_NVRAM_TARGET_ACTIVE,
BMAPI_NVRAM_TARGET_NVRAM or
BMAPI_NVRAM_TARGET_OTP.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401), NetXtreme I, NetXtreme II

Remarks:

1. Applications need to call BmapiInitDiag() prior to call this and call BmapiUnInitDiag() after everything is done.
2. 'uTarget' is applicable only to NetLink and NetXtreme.
3. BMAPI_NVRAM_TARGET_OTP is applicable only to some NetXtreme devices only.

5.145 BmapiGetReverseNWayStatus

BmapiGetReverseNWayStatus() will get Reverse nWay feature status from NVRAM.

```
U32 BmapiGetReverseNWayStatus(
  IN U32 handle,
  OUT U32 *pEnable )
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

pEnable

Pointer to a U32 number. If 0, Reverse nWay is disabled. Otherwise, Reverse nWay is enabled.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401), NetXtreme I

Remarks:

none

5.146 BmapiSetReverseNWay

BmapiSetReverseNWay() will set Reverse nWay feature to NVRAM.

```
U32 BmapiSetReverseNWay(
  IN U32 handle,
  IN U32 uEnable )
```

Parameters:

handle

Handle to an adapter BM_ADAPTER_INFO structure.

uEnable

If 0, Reverse nWay is disabled. Otherwise, Reverse nWay is enabled.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink (exclude 4401), NetXtreme I

Remarks:

none

5.147 BmapiGetNicStatisticsV3

BmapiGetNicStatisticsV3() will return statistics information for a NDIS driver.

```
U32 BmapiSetReverseNWay(
  IN U32 handle,
  IN/OUT BM_GENERAL_STATISTICS_EX *pGenStatistics,
  IN/OUT BM_ETHERNET_STATISTICS_EX *pEthStatistics )
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pGenStatistics`

Pointer to a BM_GENERAL_STATISTICS_EX structure for general statistics information of a network adapter. The data will be filled out on return.

`pEthStatistics`

Pointer to an BM_ETHERNET_STATISTICS_EX structure for ethernet statistics information of a network adapter. The data will be filled out on return.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetLink, NetXtreme I, NetXtreme II

Remarks:

1. Applications should allocate buffers for the statistics structures and pass pointers to those structures to the function. Application should be aware that some information might not be available.
2. “version” in the structure should be initialized on input.

5.148 BmapiOfldStatistics

BmapiOfldStatistics() will query offload statistics from NDIS driver.

```
U32 BmapiSetReverseNWay(
  IN U32 handle,
  IN/OUT BM_IP_OFLD_STATS *pIp4OfldStat,
  IN/OUT BM_IP_OFLD_STATS *pIp6OfldStat,
  IN/OUT BM_TCP_OFLD_STATS *pTcp4OfldStat,
  IN/OUT BM_TCP_OFLD_STATS *pTcp6OfldStat )
```

Parameters:

`handle`

Handle to an adapter BM_ADAPTER_INFO structure.

`pIp4OfldStat`

Pointer to a BM_IP_OFLD_STATS structure for the returned IPv4 offload statistics information.

`pIp6OfldStat`

Pointer to a BM_IP_OFLD_STATS structure for the returned Ipv6 offload statistics information.

`pTcp4OfldStat`

Pointer to a BM_TCP_OFLD_STATS structure for the returned TCPv4 offload statistics information.

`pTcp6OfldStat`

Pointer to a BM_TCP_OFLD_STATS structure for the returned TCPv6 offload statistics information.

Return Value:

Return BMAPI_OK if successful; otherwise return a nonzero error code.

Supported Network Adapters:

NetXtreme II

Remarks:

1. Applications should allocate buffers for the statistics structures and pass pointers to those structures to the function. Application should be aware that some information might not be available.
2. "version" in the structure should be initialized on input.
3. The statistics is available only for NDIS 5.2 or later.
4. The miniport needs to support chimney offload.

5.149 BmapiGetLldpParams

BmapiGetLldpParams() will retrieve LLDP parameters.

```
U32 BmapiGetLldpParams(
    IN U32 handle,
    IN/OUT BM_LLDP_PARAMS *pLldpParams )
```

Parameters:

handle

Handle to the NIC information that should be retrieved.

pLldpParams

Pointer to BM_LLDP_PARAMS for LLDP parameters.

Return Value:

Returns BMAPI_OK if successful; otherwise returns a nonzero error code.

Supported Network Adapters:

FCoE supported adapters

Remarks:

1. Applications should allocate buffers for the structures and pass the pointer to the structure to the function. Application should be aware that some information might not be available.
2. The 'handle' should be for a VBD.

5.150 BmapiGetDcbxParams

BmapiGetDcbxParams() will retrieve DCBX parameters.

```
U32 BmapiGetDcbxParams(
    IN U32 handle,
    IN/OUT BM_DCBX_PARAMS *pDcbxParams )
```

Parameters:

handle

Handle to the NIC information that should be retrieved.

pDcbxParams

Pointer to BM_DCBX_PARAMS for DCBX parameters.

Return Value:

Returns BMAPI_OK if successful; otherwise returns a nonzero error code.

Supported Network Adapters:

FCoE supported adapters

Remarks:

1. Applications should allocate buffers for the structures and pass the pointer to the structure to the function. Application should be aware that some information might not be available.
2. The 'handle' should be for a VBD.

5.151 BmapiGetIscsiCfg

BmapiGetIscsiCfg() will read iSCSI boot configuration block from NVRAM.

```
U32 BmapiGetIscsiCfg(
  IN U32 handle,
  IN/OUT U8 *pBuf,
  IN/OUT U32 *pLen )
```

Parameters:

`handle`

Handle to the NIC to read iSCSI boot configuration.

`pBuf`

Buffer for iSCSI boot configuration

`pLen`

Pointer to the buffer length on input and actual iSCSI configuration length on return.

Return Value:

Returns BMAPI_OK if successful; otherwise returns a nonzero error code.

Supported Network Adapters:

All adapters that have iSCSI boot programmed

Remarks:

1. If 'pBuf' is NULL, the required buffer length will be in 'pLen'. If 'pBuf' is not NULL, the iSCSI CFG block length will be in 'pLen'.
2. For NXII, the 'handle' must be a VBD handle.

5.152 BmapiSetIscsiCfg

BmapiSetIscsiCfg() will update iSCSI boot configuration block in NVRAM.

```
U32 BmapiSetIscsiCfg(
  IN U32 handle,
  IN/OUT U8 *pBuf,
  IN U32 uLen )
```

Parameters:

`handle`

Handle to the NIC to write iSCSI boot configuration.

`pBuf`

Buffer for iSCSI boot configuration

uLen

How many bytes from the 'pBuf' to write to NVRAM.

Return Value:

Returns BMAPI_OK if successful; otherwise returns a nonzero error code.

Supported Network Adapters:

All adapters that have iSCSI boot programmed

Remarks:

1. The 'uLen' must be the same length as existing iSCSI boot configuration block.
2. For NXII, the 'handle' must be a VBD handle.

5.153 BmapiGetFcoeCfg

BmapiGetFcoeCfg() will read FCoE boot configuration block from NVRAM.

```
U32 BmapiGetFcoeCfg(
  IN U32 handle,
  IN/OUT U8 *pBuf,
  IN/OUT U32 *pLen )
```

Parameters:

handle

Handle to the NIC to read iSCSI boot configuration.

pBuf

Buffer for FCoE boot configuration

pLen

Pointer to the buffer length on input and actual FCoE configuration length on return.

Return Value:

Returns BMAPI_OK if successful; otherwise returns a nonzero error code.

Supported Network Adapters:

All adapters that have FCoE boot programmed

Remarks:

1. If 'pBuf' is NULL, the required buffer length will be in 'pLen'. If 'pBuf' is not NULL, the FCoE boot configuration block length will be in 'pLen'.
2. For NXII, the 'handle' must be a VBD handle.

5.154 BmapiSetFcoeCfg

BmapiSetFcoeCfg() will update FCoE boot configuration block in NVRAM.

```
U32 BmapiSetFcoeCfg(
  IN U32 handle,
  IN/OUT U8 *pBuf,
  IN U32 uLen )
```

Parameters:

handle

Handle to the NIC to write FCoE boot configuration.

pBuf

Buffer for FCoE boot configuration

uLen

How many bytes from the 'pBuf' to write to NVRAM.

Return Value:

Returns BMAPI_OK if successful; otherwise returns a nonzero error code.

Supported Network Adapters:

All adapters that have FCoE boot programmed

Remarks:

1. The 'uLen' must be the same length as existing iSCSI boot configuration block.
2. For NXII, the 'handle' must be a VBD handle.

5.155 BmapiGetMBAParams

BmapiGetMBAParams() will retrieve MBA parameters from NVRAM.

```
U32 BmapiGetMBAParams(
  IN U32 handle,
  IN/OUT BM BM_MBA_PARAMS *pMbaParams )
```

Parameters:

handle

Handle to the NIC information that should be retrieved.

pMbaParams

Pointer to BM_MBA_PARAMS for MBA parameters.

Return Value:

Returns BMAPI_OK if successful; otherwise returns a nonzero error code.

Supported Network Adapters:

All NXI and NXII adapters

Remarks:

The 'handle' should be to a physical NIC/VBD.

5.156 BmapiSetMBAParams

BmapiSetMBAParams() will update MBA parameters in NVRAM.

```
U32 BmapiSetMBAParams(
  IN U32 handle,
  IN/OUT BM BM_MBA_PARAMS *pMbaParams )
```

Parameters:

handle

Handle to the NIC information that should be updated.

pMbaParams

Pointer to BM_MBA_PARAMS for MBA parameters.

Return Value:

Returns BMAPI_OK if successful; otherwise returns a nonzero error code.

Supported Network Adapters:

All NXI and NXII adapters

Remarks:

The 'handle' should be to a physical NIC/VBD.

6. Quick Programming Guide

6.1 How to get physical network adapter information?

Make sure BMAPI is initialized. And, do the following:

- Call BmapiGetNumPhyNic() to get the number of physical network adapters installed (with driver) in the system.
- Allocate enough buffer to host all data.
- Call BmapiGetAllPhyNic() to get all information for all physical network adapters.

or

- Call BmapiGetNumPhyNicEx() to get the number of physical network adapters in the system. In case of Windows 2000, the number will include network adapters not installed with driver.
- Allocate enough buffer that can host all handles for all NICs.
- Call BmapiGetAllPhyNicHandles() to get all handles for all NICs.
- Call BmapiGetPhyNic() to get detail information of a specific physical network adapter.

6.2 How to get handle to a physical network adapter?

Please follow section 5.1 to get each NIC's information. Handle can be found in every return data structure.

If the service name of a NIC is known to the application, the application can also call BmapiGetHandleByServiceName() to retrieve a handle to the NIC.

6.3 How to do diagnoctics?

Make sure BMAPI is initialized. And, do the following:

- Call BmapiInitDiag() (required step).
- Dependent on the test, the application may need to call BmapiSuspendDriver().
- Call Diag APIs such as BmapiTestControlRegisters(), BmapiTestMIIRegisters(), etc.
- If the application called BmapiSuspendDriver(), the application need to call BmapiResumeDriver() to resume traffic on the network adapter.
- Call BmapiUnInitDiag() (required step).

Please note that BmapiInitDiag() and diagnostic APIs must be called from the same thread.

6.4 How to do ASF?

- Make sure BMAPI is initialized.
- Call BmapiGetASFTable() to get ASF table from a NIC.

- Call BmapiSetASFTable() to write ASF table to a NIC.

6.5 How to tell which type of NIC it is?

- Use ‘nic_type’ in BM_ADAPTER_INFO.

7. Appendix

7.1 Data Structure and Definition

```
/****************************************************************************
 *  Function Return Code
 ***************************************************************************/
#define BMAPI_OK                        0
#define BMAPI_MEMALLOC                  1       // memory allocation failed
#define BMAPI_READCONFIG                2       // read configuration failed
#define BMAPI_BUFSHORT                  3       // memory buffer too short
#define BMAPI_INVALID_HANDLE            4       // invalid handle
#define BMAPI_INVALID_PARAMETER         5       // one or more of parameters
                                                // are invalid
#define BMAPI_UNSUPPORT_PLATFORM        6       // platform not supported
#define BMAPI_INVALID_DATA_FROM_DRIVER  7       // driver returned data is not
                                                // correct
#define BMAPI_INTERNAL_DATA_ERROR       8       // internal data error
#define BMAPI_INIT_COM_FAILED           9       // initialize COM failed
#define BMAPI_TEAM_NOT_ENOUGH_NIC       10      // Each team must have at least
                                                // on team member and an
                                                // standby adapter or two team
                                                // members
#define BMAPI_INVALID_PHY_NIC_HANDLE    11      // an invalid physical NIC
                                                // handle is used
#define BMAPI_INVALID_TEAM_NAME         12      // the team name is empty or
                                                // has leading space
#define BMAPI_INVALID_TEAM_MEMBER       13      // all adapters of the same
                                                // driver class must be either
                                                // all assigned to teams or all
                                                // unassigned
#define BMAPI_INVALID_VLAN_MEMBER       14      // all physical adapters in the
                                                // team MUST be able to do VLAN
                                                // tag. Currently, only
                                                // BMAPI_ALTEON and
                                                // BMAPI_BRCM5700 can suppoort
                                                // VLAN
#define BMAPI_DRIVER_NOT_INSTALLED      15      // intermediate or miniport
                                                // driver not installed
#define BMAPI_CONFIGURE_REG_FAILED      16      // write registry failed
#define BMAPI_REBOOT_BEFORE_ADD         17      // reboot machine before add
                                                // adapter
#define BMAPI_LOAD_LIBRARY_FAILED       18      // load external help library
                                                // failed
#define BMAPI_CONFIGURE_SYS_FAILED      19      // configure system error
#define BMAPI_DRIVER_NOT_LOADED         20      // driver is not loaded
#define BMAPI_DEVIO_CALL_FAILED         21      // DeviceIoControl() failed
#define BMAPI_BINDING_FAILED            22      // binding is not completed
#define BMAPI_SERVICE_EXIST             23      // service is installed, users
                                                // may need to reboot machine
#define BMAPI_SERVICE_MARK_DELETE       24      // service is marked to delete,
                                                // users may need to reboot
                                                // machine
#define BMAPI_CREATE_SERVICE_FAILED     25      // unable to create service
#define BMAPI_DELETE_SERVICE_FAILED     26      // unable to delete service
#define BMAPI_REMOVE_ROUTE_FAILED       27      // RemoveRouteFromNETBIOS()
                                                // failed
#define BMAPI_DRIVER_COMMUNICATE_ERROR  BMAPI_DEVIO_CALL_FAILED // obsolete
#define BMAPI_DEV_IN_DIAG               29      // the device is in diag mode
#define BMAPI_FUNCTION_NOT_ALLOWED      BMAPI_DEV_IN_DIAG   // obsolete
#define BMAPI_INVALID_TEAM_ID           30      // invalid team ID
#define BMAPI_UPDATE_CONFIG_FAILED      31      // update configuration failed
#define BMAPI_WRONG_DRIVER_VERSION      32      // the version of driver is not
                                                // supported by BMAPI
#define BMAPI_NO_MAC_ADDR               33      // no MAC address configured
#define BMAPI_NO_VIRTUAL_ADAPTER        34      // no virtual adapter
                                                // configured for the team
#define BMAPI_NO_WRITE_ACCESS_RIGHT     35      // no write access right
#define BMAPI_NOT_SUPPORTED_NIC         36      // the NIC is not supported
#define BMAPI_STANDBY_NOT_SUPPORTED     37      // stand by adapter is not
                                                // supported for the team type
```

```
#define BMAPI_CAN_NOT_LOCK_NETCFG       38      // failed to lock network
                                                // configuration in Win2000
#define BMAPI_BMAPI_NOT_INITIALIZED     39      // applications did not call
                                                // BmapiInitialize()
#define BMAPI_EVENT_FEATURE_INI_FAILED  40      // initializeation for event
                                                // feature failed
#define BMAPI_DUPLICATE_TEAM_NAME       41      // applications submit
                                                // duplicate team name when
                                                // calling BmapiApplyLBFOCfg
#define BMAPI_DUPLICATE_PHY_NIC         42      // applications submit
                                                // duplicate physical NIC in
                                                // team configuration when
                                                // calling BmapiApplyLBFOCfg
#define BMAPI_NO_VLAN_NAME              43      // applications submit more
                                                // than one virtual adapter
                                                // for a team and one or more
                                                // virtual adapter doesn't have
                                                // a VLAN name when calling
                                                // BmapiApplyLBFOCfg
#define BMAPI_DUPLICATE_VLAN_NAME       44      // applications submit
                                                // duplicate VLAN name in a
                                                // team when calling
                                                // BmapiApplyLBFOCfg
#define BMAPI_DUPLICATE_VLAN_ID         45      // applications submit
                                                // duplicate VLAN ID in a
                                                // team when calling
                                                // BmapiApplyLBFOCfg
#define BMAPI_INVALID_VLAN_ID           46      // application submit invalid
                                                // VLAN ID when calling
                                                // BmapiApplyLBFOCfg. Valid
                                                // VLAN ID could be from 0 to
                                                // 4094.
#define BMAPI_TOO_MANY_TAGGED_VLANS     47      // we only support up to
                                                // BMAPI_MAXIMUM_MEMBERS_VIR_TEAM - 1
                                                // tagged VLANs plus one
                                                // optional untagged VLAN
#define BMAPI_TOO_MANY_PHY_NICS         48      // too many physical NICs in
                                                // a team
#define BMAPI_TOO_MANY_VIR_NICS         49      // too many virtual NICs in
                                                // a team
#define BMAPI_DUPLICATE_TEAM_ID         50      // applications submit
                                                // duplicate team ID when
                                                // calling BmapiApplyLBFOCfg
#define BMAPI_NIC_NOT_FOUND             51      // NIC not found
#define BMAPI_NIC_NOT_IN_TEAM           52      // NIC is not in the team
#define BMAPI_SCM_LOCKED                53      // service control manager is
                                                // locked
#define BMAPI_UNSUPPORTED_VERSION       54      // unsupported version for
                                                // the data structure
#define BMAPI_UNSUPPORTED_IOCTL         55      // unsupported IOCTL
#define BMAPI_ASF_NOT_CAPABLE           56      // the NIC is not capable of
                                                // ASF feature
#define BMAPI_ASF_NOT_CONFIGURED        57      // the NIC is not configured
                                                // for ASF
#define BMAPI_EEPROM_CORRUPTED          58      // EEPROM of the NIC is
                                                // corrupted
#define BMAPI_READ_EEPROM_FAILED        59      // read EEPROM failed
#define BMAPI_ASF_TABLE_CORRUPTED       60      // ASF table in NIC is
                                                // corrupted
#define BMAPI_SUSPEND_DRIVER_FAILED     61      // failed to suspend driver
                                                // or NIC
#define BMAPI_WRITE_ASF_TABLE_FAILED    62      // obsolete
#define BMAPI_LOCK_NIC_FAILED           63      // failed to gain exclusive
                                                // access to a NIC. Try later.
#define BMAPI_DRIVER_NOT_SUSPENDED      64
#define BMAPI_REGISTER_TEST_FAILED      65
#define BMAPI_MII_REGISTER_TEST_FAILED  66
#define BMAPI_MEMORY_TEST_FAILED        67
#define BMAPI_TEST_INTERRUPT_FAILED     68
#define BMAPI_CPU_TEST_FAILED           69
#define BMAPI_UNABLE_TO_RESET_RX_CPU    70
#define BMAPI_UNABLE_TO_RESET_TX_CPU    71
#define BMAPI_RX_CPU_TEST_FAILED        72
#define BMAPI_TX_CPU_TEST_FAILED        73
#define BMAPI_RESUME_DRIVER_FAILED      74
#define BMAPI_UNABLE_TO_GET_SERVICE_NAME    75
```

```
#define BMAPI_UNABLE_TO_LOAD_INTERMEDIATE_DRIVER    76
#define BMAPI_UNABLE_TO_SET_PACKET_FILTER   77
#define BMAPI_LOOPBACK_TEST_FAILED      78
#define BMAPI_NO_DIAG_ACCESS_RIGHT      79      // application did not call
                                                // BmapiInitDiag() with
                                                // successful return
#define BMAPI_WRITE_EEPROM_FAILED       80      // write EEPROM failed
#define BMAPI_ASF_TEST_RESET_FAILED     81      // ASF reset failed
#define BMAPI_ASF_TEST_RX_EVENT_FAILED  82      // ASF RXCPU event failed
#define BMAPI_ASF_TEST_TX_EVENT_FAILED  83      // ASF TXCPU event failed
#define BMAPI_ASF_TEST_POLL_TIMER_FAILED    84  // ASF poll timer failed
#define BMAPI_ASF_TEST_ATTN_LOC_FAILED  85      // ASF ATTN loc failed
#define BMAPI_ASF_TEST_HB_TIMER_FAILED  86      // ASF heartbeat timer failed
#define BMAPI_ASF_TEST_PL_TIMER_FAILED  87      // ASF poll legacy timer failed
#define BMAPI_ASF_TEST_RT_TIMER_FAILED  88      // ASF retransmit timer failed
#define BMAPI_ASF_TEST_WD_TIMER_FAILED  89      // ASF watchdog timer failed
#define BMAPI_ASF_TEST_NO_STAMP         90      // ASF timestamp counter is not
                                                // counting
#define BMAPI_NO_WRITE_EEPROM_PRIV      91      // No priviledge to write
                                                // EEPROM
#define BMAPI_FW_NOT_LOADED             92      // Firmware is not loaded
#define BMAPI_NO_READ_EEPROM_PRIV       93      // No priviledge to read
                                                // EEPROM
#define BMAPI_NO_SYSTEM_UUID            94      // no system UUID found
#define BMAPI_NO_ASF_TABLES             95      // no ASF tables found
#define BMAPI_UNSUPPORTED_FW_VERSION    96      // unsupported firmware version
#define BMAPI_INVALID_OFFSET_OR_LEN     97      // offset or length must be at
                                                // 32-bit alignment
#define BMAPI_READ_REGISTER_FAILED      98      // read register failed
#define BMAPI_WRITE_REGISTER_FAILED     99      // write register failed
#define BMAPI_NO_READ_NIC_MEM_PRIV      100     // no priviledge to read NIC
                                                // memory
#define BMAPI_READ_NIC_MEM_FAILED       101     // read NIC memory failed
#define BMAPI_NO_WRITE_NIC_MEM_PRIV     102     // no priviledge to write NIC
                                                // memory
#define BMAPI_WRITE_NIC_MEM_FAILED      103     // write NIC memory failed
#define BMAPI_CREATELOCK_FAILED         104     // create lock fialed
#define BMAPI_READ_PCI_CONFIG_FAILED    105     // read PCI configuration
#define BMAPI_FILE_OPEN_FAILED          106     // file open failed
#define BMAPI_MEM_MAP_FAILED            107     // map mempry failed
#define BMAPI_NO_IP_INFO                108     // can not find IP information
                                                // for an adapter, could be
                                                // driver is not loaded or
                                                // no IP bind to the adapter
#define BMAPI_NETTEST_INI_FAILED        109     // NetworkTest init failed
                                                // no IP protocol???
#define BMAPI_NETTEST_CFG_FAILED        110     // NetworkTest configure test
                                                // failed
#define BMAPI_NETTEST_ADAPT_NOIP        111     // adapter has no valid IP
#define BMAPI_NETTEST_ADAPT_BIND        112     // bind to adapter failed
#define BMAPI_NETTEST_INVALID_DEST      113     // invalid destination address
#define BMAPI_NETTEST_SEND_FAILED       114     // send packet failed
#define BMAPI_NETTEST_RECV_FAILED       115     // receive packet failed
#define BMAPI_NETTEST_TEST_FAILED       116     // no response received
#define BMAPI_INVALID_TEAMTYPE_CHANGE   117     // unsupported team type
                                                // changed
#define BMAPI_INVALID_TTCFG_PHY         118     // obsolete
#define BMAPI_UNSUPPORTED_TT_OS         119     // obsolete
#define BMAPI_NO_TT_IN_BASP             120     // obsolete
#define BMAPI_NOT_TT_NIC                121     // obsolete
#define BMAPI_CONFIG_TT_FAILED          122     // obsolete
#define BMAPI_INVALID_TEAMTYPE          123     // unsupported team type
#define BMAPI_NON_TT_DRIVER             124     // obsolete
#define BMAPI_PHY_OFF                   125     // PHY is off
#define BMAPI_LOOPBACK_TEST_NOLINK      126     // no link for loopback test
#define BMAPI_OBSOLETE_API              127     // the API is now obsolete
#define BMAPI_FILE_NOT_FOUND            128     // dependent file is not found
#define BMAPI_OS_ACCESS_DENIED          129     // user does not have access
                                                // right
#define BMAPI_CONNECT_SCM_FAILED        130     // failed to connect to service
                                                // control manager
#define BMAPI_NOT_SUPPORTED_DRV         131     // the driver is not supported
#define BMAPI_FEATURE_NOT_AVAILABLE     200     // the request feature is not
                                                // available for the device
#define BMAPI_REBOOT_REQUIRED           201     // reboot machine is required
                                                // to recover from the failure
```

```
#define BMAPI_RESTART_DEVICE_FAILED     202     // failed to restart a device
#define BMAPI_CREATE_DEVICE_FAILED      203     // failed to create a device
#define BMAPI_FIND_DEVICE_FAILED        204     // failed to find an installed
                                                // device. Could be driver is
                                                // not installed yet.
#define BMAPI_LED_TEST_FAILED           205     // LED test failed
#define BMAPI_LICENSE_INFO_TIMEOUT      206     // read or write license key
                                                // timeout
#define BMAPI_LICENSE_INFO_BUSY         207     // another license information
                                                // request pending, try again
                                                // later
#define BMAPI_LICENSE_KEY_NOTEXIST      208     // the license key does not
                                                // exist
#define BMAPI_LICENSE_KEY_INVALID       209     // the license key is invalid
#define BMAPI_NO_VALID_LICENSE_KEY      210     // no valice key cen be found
                                                // in the NIC
#define BMAPI_CON_EXCEED_HW_MAX         211     // number of reserved
                                                // connections exceeds
                                                // hardware limitation
#define BMAPI_CON_EXCEED_LIC_MAX        212     // number of reserved
                                                // connections exceeds
                                                // licensed connections
#define BMAPI_RES_EXCEED_HW_MAX         213     // total reserved resources
                                                // exceeds hardware limitation
#define BMAPI_RES_NOT_LICENSED          214     // reserved connections are
                                                // not licensed
#define BMAPI_UNSUPPORTED_ADDR_TYPE     215     // unsupported address type
#define BMAPI_DUPLICATE_ADDR            216     // duplicate address
#define BMAPI_PROBE_UNSUPPORTED         217     // probe packe is not supported
#define BMAPI_NO_TARGET_IP              218     // no target IP configured for
                                                // Probe Packet feature
#define BMAPI_NO_PHY_PROBE_IP           219     // no IP configured for
                                                // physical NIC for Probe
                                                // Packet feature
#define BMAPI_CABDIAG_FAIL              220     // cable diag failed
#define BMAPI_CREATE_THREAD_FAILED      221     // failed to create thread
#define BMAPI_DOING_DIAGNOSTICS         222     // a diagnostics is running
                                                // already
#define BMAPI_NO_DIAGNOSTICS            223     // no pending diagnostics
#define BMAPI_CAN_NOT_STOP_DIAG         224     // failed to stop diagnostics
#define BMAPI_MANUF_KEY_CRC_ERR         225     // manufacturing key CRC error
#define BMAPI_UPG_KEY_CRC_ERR           226     // upgrade key CRC error
#define BMAPI_ASFMBOX_NOT_FOUND         227     // ASF mailbox is not found
#define BMAPI_INVALID_MBOX_NUM          228     // invalid ASF mailbox number
#define BMAPI_ISCSI_BOOT_DEV            229     // The NIC is on iSCSI boot.
#define BMAPI_INVALID_PEND_TASK         230     // invalid iSCSI pending task
#define BMAPI_INVALID_TEAM_TYPE_W_ISCSI 231     // invalid team type with iSCSI
#define BMAPI_APEEVT_TIMEOUT            232     // send APE event time out
#define BMAPI_APEEVT_MSG_TOO_LONG       233     // send APE event message too
                                                // long
#define BMAPI_NOT_ENOUGH_DATA           234     // request more that data
                                                // length
#define BMAPI_DATA_NOT_FOUND            235     // requested data not found
#define BMAPI_MGMT_FW_UNAVAILABLE       236     // maganement FW not running
#define BMAPI_APE_NOT_STOP              237     // APE is not stopped
#define BMAPI_DIR_FOUND                 238
#define BMAPI_DIR_FULL                  239
#define BMAPI_INVALID_NVRAM_SIZE        240
#define BMAPI_NVRAM_FULL                241
#define BMAPI_CONNECT_WMI_FAILED        242     // failed to connect to WMI
#define BMAPI_GET_WMI_OBJECT_FAILED     243     // failed to get WMI object
#define BMAPI_GET_PROPERTY_FAILED       244     // failed to get WMI object
                                                // property
#define BMAPI_UNSUPPORTED_LICENSE_VER   245     // unsupported license key
                                                // version
#define BMAPI_TOO_MANY_TEAMS            246     // configure too many teams
#define BMAPI_NO_NVMDIR_FOUND           247     // can not find a NVRAM
                                                // directory entry
#define BMAPI_UNSUPPORTED_FW            248     // unsupported FW image
#define BMAPI_NVRAM_PROGRAM_FAILED      249     // NVRAM download failed
#define BMAPI_NO_DIR                    250     // NVRAM image does not support
                                                // directory
#define BMAPI_BOOT_DEV                  251     // The port is a boot device.
```

```
/****************************************************************************
 *  Other definitions
 ****************************************************************************/
#define BMAPI_MAJOR_VERSION             7
#define BMAPI_MINOR_VERSION             19
#define BMAPI_BUILD_VERSION             0

#define BMAPI_MAX_PRODUCT_LEN           80
#define BMAPI_MAX_DESC_LEN              80
#define BMAPI_MAX_VLAN_NAME_LEN         40
#define BMAPI_MAX_TEAM_NAME_LEN         40
#define BMAPI_MAX_IP_ADDR_LEN           20
#define BMAPI_MAX_MEMORY_ADDR_LEN       80
#define BMAPI_MAX_MAC_ADDR_LEN          20
#define BMAPI_MAX_TITLE_LEN             80
#define BMAPI_MAX_MFG_LEN               80
#define BMAPI_MAX_DRV_NAME              32
#define BMAPI_MAX_DRV_INT_TYPE          16
#define BMAPI_MAX_DRVVER_LEN            40
#define BMAPI_MAX_TEAM                  8

#define BMAPI_LBFO_MINIPORT_NAME        "Blfm"
#define BMAPI_LBFO_MINIPORT_DESC        "Broadcom Ndis Miniport Driver"
#define BMAPI_LBFO_MINIPORT_TITLE       "BASP Virtual Adapter"

#define BMAPI_INVALID_NIC_HANDLE        0
#define BMAPI_INVALID_TEAM_HANDLE       0

#define BMAPI_MAX_VLAN_ID               4094

// return 'status' of BmapiLBFOSoftwareStatus()
#define BMAPI_LBFO_STATUS_NOT_INSTALLED 0
#define BMAPI_LBFO_STATUS_NOT_LOADED    1
#define BMAPI_LBFO_STATUS_LOADED        2

// Definition for BmapiEnableDevice()
#define BMAPI_NIC_DISABLE               0
#define BMAPI_NIC_ENABLE                1
#define BMAPI_NIC_PROPCHANGE            2

#define BMAPI_FW_MAX_DESCRIPTION_LEN    16

// 'uType' definition for BmapiReadNicMem() and BmapiWriteNicMem().
#define BMAPI_INDIRECT_REG_READ         0
#define BMAPI_INDIRECT_MEM_READ         1
#define BMAPI_PHY_REG_READ              2
#define BMAPI_REG_READ                  3
#define BMAPI_MEM_READ                  4

#define BMAPI_INDIRECT_REG_WRITE        0
#define BMAPI_INDIRECT_MEM_WRITE        1
#define BMAPI_PHY_REG_WRITE             2
#define BMAPI_REG_WRITE                 3
#define BMAPI_MEM_WRITE                 4

// 'uType' definition for BmapiGetIpAddrInfo().
#define BMAPI_IPINFO_IP_LIST            0
#define BMAPI_IPINFO_SUBNETMASK_LIST    1
#define BMAPI_IPINFO_GATEWAY_LIST       2

// 'uWhere' definition for BmapiGetPowerMode().
#define BMAPI_GET_POWER_MODE_DRIVER     0
#define BMAPI_GET_POWER_MODE_EEPROM     1

// 'uWhere' definition for BmapiSetPowerMode().
#define BMAPI_SET_POWER_MODE_DRIVER     0
#define BMAPI_SET_POWER_MODE_EEPROM     1
#define BMAPI_SET_POWER_MODE_BOTH       2       /* EEPROM and driver        */

// Mode definition for BmapiGetPowerMode and BmapiSetPowerMode().
#define BMAPI_POWER_MODE_FULL           0
#define BMAPI_POWER_MODE_LOW            1

// 'uOption' definition for BmapiWriteFirmwareInfo().
#define BMAPI_WR_FW_MANUFAC             0x00000001
```

```
#define BMAPI_MAX_COMMUNITY_NAME_LEN    20

// 'uPhyStatus' definition for BmapiSetPHYStatus() and BmapiGetPHYStatus().
#define BMAPI_PHY_STATUS_ON             0
#define BMAPI_PHY_STATUS_OFF            1

// Loopback type for BmapiTestLoopBack() and BmapiTestLoopBackEx()
#define BMAPI_LOOPBACK_TYPE_MAC         0
#define BMAPI_LOOPBACK_TYPE_PHY         1
#define BMAPI_LOOPBACK_TYPE_TWO_NODE    2
#define BMAPI_LOOPBACK_TYPE_EXTERNAL    3

// Target type for BmapiReadFirmware2()
#define BMAPI_NVRAM_TARGET_ACTIVE       0   // The device currently is using.
#define BMAPI_NVRAM_TARGET_NVRAM        1   // SEEPROM or flash
#define BMAPI_NVRAM_TARGET_OTP          2   // OTP

/****************************************************************************
 *  OBSOLETE loopback definition
 ****************************************************************************/
//#define LOOPBACK_TYPE_MAC               BMAPI_LOOPBACK_TYPE_MAC
//#define LOOPBACK_TYPE_PHY               BMAPI_LOOPBACK_TYPE_PHY
//#define LOOPBACK_TYPE_TWO_NODE          BMAPI_LOOPBACK_TYPE_TWO_NODE



/****************************************************************************
 *  OBSOLETE starts here
 ****************************************************************************/
// Used by BmapiGetMultiBRCMNicParams()
#define BMAPI_BRCMNIC_PARAM_TYPE_UNKNOWN    0
#define BMAPI_BRCMNIC_PARAM_TYPE_DWORD      1
#define BMAPI_BRCMNIC_PARAM_TYPE_BINARY     2   // binary data

#define BMAPI_BRCMNIC_PARAM_RESULT_OK           0
#define BMAPI_BRCMNIC_PARAM_RESULT_NOTEXIST     1   // not exist
#define BMAPI_BRCMNIC_PARAM_RESULT_NOTSUPPORTED 2   // not supported
#define BMAPI_BRCMNIC_PARAM_RESULT_ERROR        3   // registry reading error
#define BMAPI_BRCMNIC_PARAM_RESULT_BUFTOOSHORT  4   // buffer too short
#define BMAPI_BRCMNIC_PARAM_RESULT_WRONGDATALEN 5   // incorrect parameter data length
#define  BMAPI_BRCMNIC_PARAM_RESULT_DATAINVALID    6     //  invalid  data  value  for  the
parameter

// Configurable Broadcom NIC parameters.
#define BMAPI_BRCMNIC_RXJUMBODESCCNT    0   // type: DWORD, range: 0-255, default: 0
#define BMAPI_BRCMNIC_TASKOFFLOADCAP    1   // type: DWORD
                                            // value: 0 - None (default)
                                            // value: 21 - Tx TCP/IP Checksum
                                            // value: 42 - Rx TCP/IP Checksum
                                            // value: 63 - Tx/Rx TCP/IP Checksum
#define BMAPI_BRCMNIC_WAKEUPMODECAP     2   // type: DWORD
                                            // value: 0 - None
                                            // value: 1 - Magic Packet
                                            // value: 2 - Wake Up Frame
                                            // value: 3 - Both (default)
#define BMAPI_BRCMNIC_FLOWCONTROLCAP    3   // type: DWORD
                                            // value: 0 - Disable (default)
                                            // value: 1 - Rx PAUSE
                                            // value: 2 - Tx PAUSE
                                            // value: 3 - Rx/Tx PAUSE
                                            // value: 2147483648 - Auto
#define BMAPI_BRCMNIC_ENABLE8021P       4   // type: DWORD
                                            // value: 0 - Disable (default)
                                            // value: 1 - Enable
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE    5   // type: DWORD
                                            // value: 1500 (bytes) (default)
                                            // value: 2000 (bytes)
                                            // value: 2500 (bytes)
                                            // value: 3000 (bytes)
                                            // value: 3500 (bytes)
                                            // value: 4000 (bytes)
                                            // value: 4500 (bytes)
                                            // value: 5000 (bytes)
                                            // value: 5500 (bytes)
                                            // value: 6000 (bytes)
                                            // value: 6500 (bytes)
                                            // value: 7000 (bytes)
                                            // value: 7500 (bytes)
                                            // value: 8000 (bytes)
#define BMAPI_BRCMNIC_NETWORKADDRESS    6   // type: binary (6 bytes)
#define BMAPI_BRCMNIC_WOL_SPEED         7   // type: DWORD
                                            // value: 0 - Auto (default)
                                            // value: 1 - 10 Mb
                                            // value: 2 - 100 Mb

// Parameter values for Broadcom NIC.
#define BMAPI_BRCMNIC_RXJUMBODESCCNT_MIN                0   // default
#define BMAPI_BRCMNIC_RXJUMBODESCCNT_MAX                255

#define BMAPI_BRCMNIC_TASKOFFLOADCAP_NONE               0   // default
#define BMAPI_BRCMNIC_TASKOFFLOADCAP_TXTCPIPCHKSUM      21
#define BMAPI_BRCMNIC_TASKOFFLOADCAP_RXTCPIPCHKSUM      42
#define BMAPI_BRCMNIC_TASKOFFLOADCAP_TXRXTCPIPCHKSUM    63

#define BMAPI_BRCMNIC_WAKEUPMODECAP_NONE                0
#define BMAPI_BRCMNIC_WAKEUPMODECAP_MAGICPACKET         1
#define BMAPI_BRCMNIC_WAKEUPMODECAP_WAKEUPFRAME         2
#define BMAPI_BRCMNIC_WAKEUPMODECAP_BOTH                3   // default

#define BMAPI_BRCMNIC_FLOWCONTROLCAP_DISABLE            0   // default
#define BMAPI_BRCMNIC_FLOWCONTROLCAP_RXPAUSE            1
#define BMAPI_BRCMNIC_FLOWCONTROLCAP_TXPAUSE            2
#define BMAPI_BRCMNIC_FLOWCONTROLCAP_RXTXPAUSE          3
#define BMAPI_BRCMNIC_FLOWCONTROLCAP_AUTO               2147483648

#define BMAPI_BRCMNIC_ENABLE8021P_DISABLE               0   // default
#define BMAPI_BRCMNIC_ENABLE8021P_ENABLE                1

#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_1500               1500    // default
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_2000               2000
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_2500               2500
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_3000               3000
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_3500               3500
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_4000               4000
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_4500               4500
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_5000               5000
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_5500               5500
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_6000               6000
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_6500               6500
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_7000               7000
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_7500               7500
#define BMAPI_BRCMNIC_JUMBOFRAMESIZE_8000               8000

#define BMAPI_BRCMNIC_WOL_SPEED_AUTO                    0   // default
#define BMAPI_BRCMNIC_WOL_SPEED_10MB                    1
#define BMAPI_BRCMNIC_WOL_SPEED_100MB                   2
/*****************************************************************************
 *  OBSOLETE ends here
 *****************************************************************************/



#ifndef BOOL
typedef int                 BOOL;
#endif

#pragma pack(push, 1)

/*****************************************************************************
 *  Event related definitions.
 *****************************************************************************/
//
// extra_info for EVT_ACTIVE and EVT_INACTIVE
//
#define BMAPI_EVINFO_ACTIVITY_LINK      0   // active/inactive due to link status
#define BMAPI_EVINFO_ACTIVITY_MANUAL    1   // active/inactive due to enable/disable
#define BMAPI_EVINFO_ACTIVITY_ADDREMOVE 2   // active/inactive due to add/remove
#define BMAPI_EVINFO_ACTIVITY_STANDBY   3   // active/inactive due to standby
#define BMAPI_EVINFO_ACTIVITY_NONE      0xFFFFFFFF

typedef enum
{
    BMAPI_EVT_ACTIVE = 0,           // adapter is active
    BMAPI_EVT_INACTIVE = 1,         // adapter is inactive
    BMAPI_EVT_LINKUP = 2,           // adapter's link status goes up
    BMAPI_EVT_LINKDOWN = 3,         // adapter's link status goes down
    BMAPI_EVT_ENABLE = 4,           // adapter is enabled via management interface
    BMAPI_EVT_DISABLE = 5,          // adapter is disabled via management interface
    BMAPI_EVT_ADD = 6,              // adapter is added to a team
    BMAPI_EVT_REMOVE = 7,           // adapter is removed from a team
    BMAPI_EVT_ASFCFG_CHG = 100,     // adapter's ASF configuration is modified
    BMAPI_EVT_DEV_ARRIVED = 101,    // a device is arrived or enabled
    BMAPI_EVT_DEV_REMOVED = 102,    // a device is to be removed

#if defined(__NETWARE__)
    BMAPI_EVT_DUMMY = 0xFFFFFFF     // make it a 32-bit variable
#else
    BMAPI_EVT_DUMMY = 0xFFFFFFFF    // make it a 32-bit variable
#endif
} BMAPI_EV_TYPE;


#define BMAPI_EVT_DATA_VER       1

typedef struct
{
    //  Version is defined as BMAPI_EVT_DATA_VER.
    //  'version' is always specify on callback
    U32                 version;

} BMAPI_EVT_DATA;

typedef void (*BMAPIEVENTCALLBACK)( U32 event, const S8 *team_name, U32 team_id, const S8
*adapter_service_name, U32 adapter_handle, U32 lb_number, U32 sb_number, U32 extra_info,
void *cookie );




/****************************************************************************
 *  Data Structures
 ****************************************************************************/
// These are for 'nic_type' phisical devices.
#define BMAPI_NON_BROADCOM_NIC          0
#define BMAPI_ALTEON                    1
#define BMAPI_BRCM5700                  2
#define BMAPI_UNKNOWN_NIC               3       // driver is not running
#define BMAPI_BRCM5706                  4
#define BMAPI_BRCM57710                 5

// These are the 'nic_type' for virtual devices.
#define BMAPI_BASP_VIR                  100
#define BMAPI_VIR_NDIS                  101
#define BMAPI_VIR_WSD                   102
#define BMAPI_VIR_ISCSI                 103
#define BMAPI_VIR_DIAG                  104
#define BMAPI_VIR_FCOE                  105
#define BMAPI_5706_NDIS                 BMAPI_VIR_NDIS
#define BMAPI_5706_WSD                  BMAPI_VIR_WSD
#define BMAPI_5706_ISCSI                BMAPI_VIR_ISCSI
#define BMAPI_5706_DIAG                 BMAPI_VIR_DIAG

// 'member_type' in BM_ADAPTER_INFO
#define BMAPI_MEMBER_ROLE_LOAD_BALANCE  0
#define BMAPI_MEMBER_ROLE_STAND_BY      1

// 'state' in BM_ADAPTER_INFO
#define BMAPI_MEMBER_STATE_UP           0
#define BMAPI_MEMBER_STATE_DOWN         1
#define BMAPI_MEMBER_STATE_DISABLED     2

typedef struct _BM_ADAPTER_INFO
{
    // NIC card type. Valid only for physical adapters.
    // BMAPI_NON_BROADCOM_NIC == non Broadcom NIC,
    // BMAPI_ALTEON == T2, BMAPI_BRCM5700 == T3, ...
    U32 nic_type;
    // Application use this handle number to indicate which adapter
    // it is referring to. If handle == BMAPI_INVALID_NIC_HANDLE,
    // The adapter is not found in current configuration.
    U32 handle;

    // pTitle will be the unique "friendly name" displayed to users.
    S8  title[BMAPI_MAX_TITLE_LEN];

    // Product name of the adapter. It Is used to IdentIfy whether two
    // adapters belong the same class of driver.
    S8  product_name[BMAPI_MAX_PRODUCT_LEN];

    // Physical adapter type in a team whether is part of a load
    // balancing team or a fail over adapter.
    // BMAPI_MEMBER_ROLE_LOAD_BALANCE == use for load balancing or unassigned.
    // BMAPI_MEMBER_ROLE_STAND_BY == fail over adapter (standby).
    U32 member_type;

    // Description of the adapter. May not be available for all adapters.
    S8  description[BMAPI_MAX_DESC_LEN];

    // Boolean flag to denote if DHCP is enabled.
    // Not available in non Windows platforms.
    U32 dhcp_enabled;

    // IP address is "192.168.0.1" format. (REG_MULTI_SZ or REG_SZ)
    S8  ip_addr[BMAPI_MAX_IP_ADDR_LEN];

    // subnet mask is "255.255.0.0" format. (REG_MULTI_SZ or REG_SZ)
    S8  subnet_mask[BMAPI_MAX_IP_ADDR_LEN];

    // defaule gateway is "192.168.0.1" format. (REG_MULTI_SZ or REG_SZ)
    S8  default_gateway[BMAPI_MAX_IP_ADDR_LEN];

    // Boolean flag to indicate whether driver is running.
    U32 driver_loaded;

    // These information is valid only driver is running.
    S8  current_mac_addr[BMAPI_MAX_MAC_ADDR_LEN];
    S8  permanent_mac_addr[BMAPI_MAX_MAC_ADDR_LEN];
    U32 major_version_number;
    U32 minor_version_number;

    // VLAN ID should be filled here. VLAN ID should greater or equal
    // to 1. Exist only for virtual adapters.
    U32 vlan_id;
    S8  vlan_name[BMAPI_MAX_VLAN_NAME_LEN];

    // State of this adapter, can be either up, down or disabled. This
    // is valid only when data is retrieved from BmapiGetTeamSnapShot().
    // Possible value:
    // BMAPI_MEMBER_STATE_UP, BMAPI_MEMBER_STATE_DOWN,
    // BMAPI_MEMBER_STATE_DISABLED
    U32 state;

    // Link speed of the physical NIC in Mbps, i.e. 10 = 10Mbps. This
    // is valid only when physical NIC card data is retrieved from
    // BmapiGetTeamSnapShot()
    U32 link_speed;

} BM_ADAPTER_INFO;



typedef struct _BM_OFFLOAD_TCP_IP_CHECKSUM
{
    // boolean value
    // not valid if the field is set to false (0).
    U32             valid;

    union {
        struct
        {
            U32         IpOptionsSupported:1;
            U32         TcpOptionsSupported:1;
            U32         TcpChecksum:1;
            U32         UdpChecksum:1;
            U32         IpChecksum:1;
        } V4Transmit;

        U32     V4Transmit_flags;
    };

    union {
        struct
        {
            U32         IpOptionsSupported:1;
            U32         TcpOptionsSupported:1;
            U32         TcpChecksum:1;
            U32         UdpChecksum:1;
            U32         IpChecksum:1;
        } V4Receive;

        U32     V4Receive_flags;
    };

    union {
        struct
        {
            U32         IpOptionsSupported:1;
            U32         TcpOptionsSupported:1;
            U32         TcpChecksum:1;
            U32         UdpChecksum:1;

        } V6Transmit;

        U32     V6Transmit_flags;
    };

    union {
        struct
        {
            U32         IpOptionsSupported:1;
            U32         TcpOptionsSupported:1;
            U32         TcpChecksum:1;
            U32         UdpChecksum:1;

        } V6Receive;

        U32     V6Receive_flags;
    };

} BM_OFFLOAD_TCP_IP_CHECKSUM;


typedef struct _BM_OFFLOAD_TCP_LARGE_SEND
{
    // boolean value
    // not valid if the field is set to false (0).
    U32     valid;

    U32     MaxOffLoadSize;
    U32     MinSegmentCount;
    U32     TcpOptions; // boolean value
    U32     IpOptions;  // boolean value

} BM_OFFLOAD_TCP_LARGE_SEND;



typedef struct _BM_OFFLOAD_TCP_CONNECTION
{
    // boolean value
    // not valid if the field is set to false (0).
    U32     valid;

    // These are boolean values. '1' means supported. '0' means
    // not supported.
    // Any of the four fields is set, the NIC is TCP offload capable.
    U32     SupportIp4;
    U32     SupportIp6;
    U32     SupportIp6ExtensionHeaders;
    U32     SupportSack;

    U32     TcpConnectionOffloadCapacity;

} BM_OFFLOAD_TCP_CONNECTION;



typedef enum
{
    BmapiIpPrefixOriginOther = 0,           // The IPv6 prefix was provided by
                                            // a source other than those
                                            // defined in this enumeration.
    BmapiIpPrefixOriginManual,              // The IPv6 prefix was manually
                                            // specified.
    BmapiIpPrefixOriginWellKnown,           // The IPv6 prefix is from a well
                                            // known source.
    BmapiIpPrefixOriginDhcp,                // The IPv6 prefix was provided by
                                            // DHCP settings.
    BmapiIpPrefixOriginRouterAdvertisement, // The IPv6 prefix was obtained
                                            // through a router advertisement
                                            // (RA).
} BM_IP_PREFIX_ORIGIN;

typedef enum
{
    BmapiIpSuffixOriginOther = 0,           // The IPv6 suffix was provided by
                                            // a source other than those
                                            // defined in this enumeration.
    BmapiIpSuffixOriginManual,              // The IPv6 suffix was manually
                                            // specified.
    BmapiIpSuffixOriginWellKnown,           // The IPv6 suffix is from a well
                                            // known source.
    BmapiIpSuffixOriginDhcp,                // The IPv6 suffix was provided by
                                            // DHCP settings.
    BmapiIpSuffixOriginLinkLayerAddress,    // The IPv6 suffix was obtained
                                            // from the link-layer address.
    BmapiIpSuffixOriginRandom,              // The IPv6 suffix was obtained
                                            // from a random source.
} BM_IP_SUFFIX_ORIGIN;

typedef enum
{
    BmapiIpDadStateInvalid = 0,             // The DAD state is invalid.
    BmapiIpDadStateTentative,               // The DAD state is tentative.
    BmapiIpDadStateDuplicate,               // A duplicate IPv6 address has
                                            // been detected.
    BmapiIpDadStateDeprecated,              // The IPv6 address has been
                                            // deprecated.
    BmapiIpDadStatePreferred,               // The IPv6 address is the
                                            // preferred address.
} BM_IP_DAD_STATE;


#define BMAPI_IP_ADDR_DNS_ELIGIBLE   0x01   // The address is legal to appear
                                            // in DNS.
#define BMAPI_IP_ADDR_TRANSIENT      0x02   // The address is a cluster address
                                            // and should not be used by most
                                            // applications.

typedef struct _BM_IP_ADDRESS
{
    // The first two fields should be equivelent to:
    // struct sockaddr_in6
    // {
    //        short   sin6_family;
    //        u_short sin6_port;
    //        u_long  sin6_flowinfo;
    //        struct  in6_addr sin6_addr;
    //        u_long  sin6_scope_id;
    // };
    U16     sa_family;              // address family
    char    sa_data[26];
} BM_IP_ADDRESS;
```

```
typedef struct _BM_IP_UNICAST_ADDRESS
{
    // Boolean value. If 0, the data are invalid and shuld NOT be used.
    U32                 valid;

    // Currently, the following values are supported:
    // BMAPI_IP_ADDR_DNS_ELIGIBLE
    // BMAPI_IP_ADDR_TRANSIENT
    U32                 Flags;

    // IP address
    BM_IP_ADDRESS       Address;

    // Only available for Vista or later.
    U8                  PrefixLength;

    U8                  reserved1[35];

    BM_IP_PREFIX_ORIGIN PrefixOrigin;
    BM_IP_SUFFIX_ORIGIN SuffixOrigin;
    BM_IP_DAD_STATE     DadState;           // Duplicate Address Detection
                                            // (DAD) state.

    U32                 ValidLifetime;      // Valid lifetime for the address,
                                            // in seconds.
    U32                 PreferredLifetime;  // Preferred lifetime for the
                                            // address, in seconds.
    U32                 LeaseLifetime;      // Lease lifetime for the address,
                                            // in seconds.
    U8                  reserved2[32];
} BM_IP_UNICAST_ADDRESS;



typedef struct _BM_IP_ADAPTER_GATEWAY_ADDRESS {
    // Boolean value. If 0, the data are invalid and shuld NOT be used.
    U32                 valid;

    BM_IP_ADDRESS       Address;
} BM_IP_ADAPTER_GATEWAY_ADDRESS;



#define BM_OFFLOAD_NOT_SUPPORTED            0
#define BM_OFFLOAD_SUPPORTED                1

#define BM_OFFLOAD_SET_NO_CHANGE            0
#define BM_OFFLOAD_SET_ON                   1
#define BM_OFFLOAD_SET_OFF                  2

//
// Encapsulation types that are used during offload in query
//
#define BM_ENCAPSULATION_NOT_SUPPORTED              0x00000000
#define BM_ENCAPSULATION_NULL                       0x00000001
#define BM_ENCAPSULATION_IEEE_802_3                 0x00000002
#define BM_ENCAPSULATION_IEEE_802_3_P_AND_Q         0x00000004
#define BM_ENCAPSULATION_IEEE_802_3_P_AND_Q_IN_OOB  0x00000008
#define BM_ENCAPSULATION_IEEE_LLC_SNAP_ROUTED       0x00000010

typedef struct _BM_TCP_LARGE_SEND_OFFLOAD_V1
{

    struct
    {
        U32     Encapsulation;
        U32     MaxOffLoadSize;
        U32     MinSegmentCount;
        U32     TcpOptions;
        U32     IpOptions;
    } IPv4;

} BM_TCP_LARGE_SEND_OFFLOAD_V1;

typedef struct _BM_TCP_IP_CHECKSUM_OFFLOAD
```

```
{

    struct
    {
        U32     Encapsulation;
        U32     IpOptionsSupported;
        U32     TcpOptionsSupported;
        U32     TcpChecksum;
        U32     UdpChecksum;
        U32     IpChecksum;
    } IPv4Transmit;

    struct
    {
        U32     Encapsulation;
        U32     IpOptionsSupported;
        U32     TcpOptionsSupported;
        U32     TcpChecksum;
        U32     UdpChecksum;
        U32     IpChecksum;
    } IPv4Receive;


    struct
    {
        U32     Encapsulation;
        U32     IpExtensionHeadersSupported;
        U32     TcpOptionsSupported;
        U32     TcpChecksum;
        U32     UdpChecksum;

    } IPv6Transmit;

    struct
    {
        U32     Encapsulation;
        U32     IpExtensionHeadersSupported;
        U32     TcpOptionsSupported;
        U32     TcpChecksum;
        U32     UdpChecksum;

    } IPv6Receive;

} BM_TCP_IP_CHECKSUM_OFFLOAD;

typedef struct _BM_IPSEC_OFFLOAD_V1
{
    struct
    {
        U32     Encapsulation;
        U32     AhEspCombined;
        U32     TransportTunnelCombined;
        U32     IPv4Options;
        U32     Flags;
    } Supported;

    struct
    {
        U32     Md5;
        U32     Sha_1;
        U32     Transport;
        U32     Tunnel;
        U32     Send;
        U32     Receive;
    } IPv4AH;

    struct
    {
        U32     Des;
        U32     Reserved;
        U32     TripleDes;
        U32     NullEsp;
        U32     Transport;
        U32     Tunnel;
        U32     Send;
        U32     Receive;
    } IPv4ESP;

} BM_IPSEC_OFFLOAD_V1;

typedef struct _BM_TCP_LARGE_SEND_OFFLOAD_V2
{
    struct
    {
         U32    Encapsulation;
         U32    MaxOffLoadSize;
         U32    MinSegmentCount;
    }IPv4;

    struct
    {
         U32    Encapsulation;
         U32    MaxOffLoadSize;
         U32    MinSegmentCount;
         U32    IpExtensionHeadersSupported;
         U32    TcpOptionsSupported;
    }IPv6;

} BM_TCP_LARGE_SEND_OFFLOAD_V2;


typedef struct _BM_TASK_OFFLOAD
{
    // Boolean value. If 0, the data are invalid and shuld NOT be used.
    U32                             valid;

    //
    // Checksum Offload information
    //
    BM_TCP_IP_CHECKSUM_OFFLOAD      Checksum;

    //
    // Large Send Offload information
    //
    BM_TCP_LARGE_SEND_OFFLOAD_V1    LsoV1;

    //
    // IPsec Offload Information
    //
    BM_IPSEC_OFFLOAD_V1             IPsecV1;
    //
    // Large Send Offload version 2Information
    //
    BM_TCP_LARGE_SEND_OFFLOAD_V2    LsoV2;

    // reserved
    U32                             Flags;

} BM_TASK_OFFLOAD;



typedef struct _BM_TCP_CONNECTION_OFFLOAD
{
    // Boolean value. If 0, the data are invalid and shuld NOT be used.
    U32     valid;

    U32     Encapsulation;
    U32     SupportIPv4;
    U32     SupportIPv6;
    U32     SupportIPv6ExtensionHeaders;
    U32     SupportSack;
    U32     TcpConnectionOffloadCapacity;
    U32     Flags;
} BM_TCP_CONNECTION_OFFLOAD;



// 'flags' in BM_ADAPTER_INFO_EX
#define BMAPI_NIC_NO_MINOPORT_DRV       0x00000001
#define BMAPI_NIC_ASF_POSSIBLE          0x00000002
```

```
// 'team_type' in BM_ADAPTER_INFO_EX, BM_VIR_NIC_INFO_EX, BM_TEAM_INFO
// and BM_TEAM_INFO2
#define BMAPI_TEAM_BRCM_LBFO            0
#define BMAPI_TEAM_FEC_GEC              1
#define BMAPI_TEAM_802_3_AD             2
#define BMAPI_TEAM_SLB_AFD              4
#define BMAPI_INVALID_TEAM_TYPE         0xFFFFFFFF

// 'link_status' in BM_ADAPTER_INFO_EX, BM_VIR_NIC_INFO_EX
// and BM_BRCM_ADAPTER_INFO
#define BMAPI_LM_STATUS_LINK_ACTIVE     4
#define BMAPI_LM_STATUS_LINK_DOWN       5

// 'media_type' in BM_ADAPTER_INFO_EX and BM_BRCM_ADAPTER_INFO
#define BMAPI_LM_MEDIA_TYPE_UNKNOWN     ( U32 )-1
#define BMAPI_LM_MEDIA_TYPE_AUTO        0
#define BMAPI_LM_MEDIA_TYPE_UTP         1
#define BMAPI_LM_MEDIA_TYPE_BNC         2
#define BMAPI_LM_MEDIA_TYPE_AUI         3
#define BMAPI_LM_MEDIA_TYPE_FIBER       4
#define BMAPI_LM_MEDIA_TYPE_SERDES          5
#define BMAPI_LM_MEDIA_TYPE_SERDES_SGMII    6
#define BMAPI_LM_MEDIA_TYPE_XGXS            7
#define BMAPI_LM_MEDIA_TYPE_XGXS_SGMII      8

// 'line_speed' in BM_ADAPTER_INFO_EX and BM_BRCM_ADAPTER_INFO
// 'link_speed' in BM_LINK_STATUS
#define BMAPI_LM_LINE_SPEED_UNKNOWN     0
#define BMAPI_LM_LINE_SPEED_10MBPS      1
#define BMAPI_LM_LINE_SPEED_100MBPS     2
#define BMAPI_LM_LINE_SPEED_1000MBPS    3
#define BMAPI_LM_LINE_SPEED_2500MBPS    4
#define BMAPI_LM_LINE_SPEED_10GBPS      5

// 'bus_type' in BM_ADAPTER_INFO_EX
#define BMAPI_BUS_TYPE_UNKNOWN          0
#define BMAPI_BUS_TYPE_UNSUPPORTED      1
#define BMAPI_BUS_TYPE_PCI              2
#define BMAPI_BUS_TYPE_PCMCIA           3
#define BMAPI_BUS_TYPE_1394             4
#define BMAPI_BUS_TYPE_USB              5

// 'arch_type' in BM_ADAPTER_INFO_EX
#define BMAPI_ARCH_VBD                  1
#define BMAPI_ARCH_L2_ONLY              2
#define BMAPI_ARCH_5706_VBD             BMAPI_ARCH_VBD
#define BMAPI_ARCH_5706_L2_ONLY         BMAPI_ARCH_L2_ONLY



#define BMAPI_ADAPTER_INFO_EX_VER       16

typedef struct _BM_ADAPTER_INFO_EX
{
    //  Version is defined as BMAPI_ADAPTER_INFO_EX_VER.
    //  'version' is required upon input
    U32                 version;

    U32                 flags;
    BM_ADAPTER_INFO     adap_info;

    // If PCI IDs information is not available, the value will be -1 and
    // 'manufacturer' will be "".
    U32                 vendor_id;
    U32                 device_id;
    U32                 subsystem_vendor_id;
    U32                 subsystem_id;
    U8                  manufacturer[BMAPI_MAX_MFG_LEN];

    // revision from PCI bus registers 0x08
    U32                 revision;

    // PCI bus information. If these information are not available,
    // data will be set to -1.
    U32                 bus_no;
    U32                 device_no;
```

```
U32                 function_no;

// PCI slot number labeled next to motherboard. This is an optional
// field (at least, LOM will not have it). If it is not available,
// this field will be set to -1.
U32                 ui_number;

// PCI information
// Valid only when vendor_id equal to 0x14E4 and
// nic_type is BMAPI_BRCM5700.
U32                 mem_base_low;
U32                 mem_base_high;

// PCI information
U32                 irq;

// NIC information
// Available only when vendor_id equal to 0x14E4 and
// nic_type is BMAPI_BRCM5700.
U32                 phy_id;
U32                 phy_addr;

// For nic_type is BMAPI_BRCM5700.
//  ChipId:4, ChipRev:4, MetalRev:8
// For example :
// 5700 A1 : 0x7000 : chip ID 0x7, chip rev 0x0, metal rev 0x01
// 5700 B2 : 0x7000 : chip ID 0x7, chip rev 0x1, metal rev 0x02
// 5701 A3 : 0x0003 : chip ID 0x0, chip rev 0x0, metal rev 0x03
//
// For nic_type is BMAPI_BRCM5706.
//  ChipId:16, ChipRev:4, MetalRev:8
U32                 chip_rev_id;

U32                 media_type;

// Possible value:
// BMAPI_LM_LINE_SPEED_UNKNOWN, BMAPI_LM_LINE_SPEED_10MBPS,
// BMAPI_LM_LINE_SPEED_100MBPS, BMAPI_LM_LINE_SPEED_1000MBPS
U32                 line_speed;

// Possible value:
// BMAPI_LM_STATUS_LINK_ACTIVE, BMAPI_LM_STATUS_LINK_DOWN
U32                 link_status;

// NIC information
// Available only when vendor_id equal to 0x14E4.
U32                 upper_misc_host_ctrl_reg;

// Driver information
U8                  driver_name[BMAPI_MAX_DRV_NAME];
U32                 driver_size;
U8                  driver_interface_type[BMAPI_MAX_DRV_INT_TYPE];
U32                 driver_interface_version_major;
U32                 driver_interface_version_minor;

// NIC information
// Available only when vendor_id equal to 0x14E4 and
// nic_type is BMAPI_BRCM5700.
U32                 internal_ram_size;

// Could be BMAPI_BUS_TYPE_UNKNOWN, BMAPI_BUS_TYPE_UNSUPPORTED,
// BMAPI_BUS_TYPE_PCI or BMAPI_BUS_TYPE_PCMCIA
U32                 bus_type;

// Team name.
S8                  team_name[BMAPI_MAX_TEAM_NAME_LEN];

// Team ID
U32                 team_id;

// The unit value is Kbps. The actual speed value will be truncated
// to the lower bound of Kbps. For example, if the speed is 14.4 Kbps,
// the value will be 14. If the speed is less than 1 Kbps, the value
// will be 0.
U32                 line_speed_Kbps;
```

```
    // Team Type. BMAPI_TEAM_BRCM_LBFO == Broadcom LBFO,
    // BMAPI_TEAM_802_3_AD == 802.3ad, BMAPI_TEAM_FEC_GEC == FEC/GEC
    // BMAPI_TEAM_SLB_AFD == SLB (Auto Fallback Disabled)
    U32                 team_type;

    // Available only when vendor_id equal to 0x14E4.
    // For 5706, it could be BMAPI_ARCH_5706_VBD.
    U32                 arch_type;

    // For 5706 VBD only. 5706 VBD will have major, minor and build version
    // numbers. major and minor version numver will be in BM_ADAPTER_INFO.
    // This will be the build number.
    U32                 drv_build_ver_num;

    // For nic_type BMAPI_BRCM5706 only.
    // PCI config space BAR size.
    U32                 bar_size;

    // For nic_type BMAPI_BRCM57710, there will be two BARs per port.
    // Here is the second BAR information.
    U32                 mem_base_low2;
    U32                 mem_base_high2;
    U32                 bar_size2;

    // For VBD and NDIS devices. Mmonolithic driver is not supported.
    U32                 mtu;

    U32                 reserved;

    // 802.3 offload capabilities
    // These three fields valid ONLY for OS prior to (include) Windows 2003.
    // For NDIS 6.0 and later (Vista and Longhorn), use 'ol_task' and
    // 'ol_conn'.
    BM_OFFLOAD_TCP_IP_CHECKSUM  ol_802_3_tcpip_chksum;
    BM_OFFLOAD_TCP_LARGE_SEND   ol_802_3_lso;
    BM_OFFLOAD_TCP_CONNECTION   ol_802_3_tcp_conn;

    // First IPv6 address found for the NIC.
    // Applications can cast like "( LPSOCKADDR )&ipv6.Address" to API
    // such as WSAAddressToString().
    BM_IP_UNICAST_ADDRESS       ipv6;

    // These two fields valid only for NDIS 6.0 and later (Vista and
    // Longhorn).
    BM_TASK_OFFLOAD             ol_task;
    BM_TCP_CONNECTION_OFFLOAD   ol_conn;

    // These handles are valid when 'arch_type' is BMAPI_ARCH_VBD.
    U32                 ndis_handle;
    U32                 wsd_handle;
    U32                 iscsi_handle;
    U32                 diag_handle;

    // First IPv6 Gateway address found for the NIC.
    // Only available for Vista or later.
    // Applications can cast like "( LPSOCKADDR )&gateway_v6.Address" to API
    // such as WSAAddressToString().
    BM_IP_ADAPTER_GATEWAY_ADDRESS   gateway_v6;

// 'ipv6_flags' in BM_ADAPTER_INFO_EX
#define BMAPI_IP_ADAPTER_DHCP_ENABLED   0x00000004
    U32                 ipv6_flags;

    // Available for NDIS 6.0 and later.
    // The link speed is in bits per second. A value of -1 in this member
    // indicates that the link speed is unknown.
    U64                 max_xmit_link_speed;
    U64                 max_rcv_link_speed;
    U64                 xmit_link_speed;
    U64                 rcv_link_speed;

    // Available for NDIS 6.0 and later.
    // Same as 'duplex_mode' in BM_BRCM_ADAPTER_INFO and BM_LINK_STATUS
    U32                 duplex_mode;

    // Available for NDIS 6.0 and later.
    // Definitions for 'rss_caps'.
#define BMAPI_RSS_CAPS_MESSAGE_SIGNALED_INTERRUPTS      0x01000000
#define BMAPI_RSS_CAPS_CLASSIFICATION_AT_ISR            0x02000000
#define BMAPI_RSS_CAPS_CLASSIFICATION_AT_DPC            0x04000000
#define BMAPI_RSS_CAPS_HASH_TYPE_TCP_IPV4               0x00000100
#define BMAPI_RSS_CAPS_HASH_TYPE_TCP_IPV6               0x00000200
#define BMAPI_RSS_CAPS_HASH_TYPE_TCP_IPV6_EX            0x00000400
#define BMAPI_RSS_HASHFUNCTIONTOEPLITZ                       0x00000001 // supported hash
function 1 -- Main RSS hash function
    U32                 rss_caps;
    U32                 rss_num_interrupt_msg;
    U32                 rss_num_rcv_que;

    // The handle is valid when 'arch_type' is BMAPI_ARCH_VBD.
    U32                 fcoe_handle;

} BM_ADAPTER_INFO_EX;



// This structure is for Broadcom BASP virtual adapter only.
#define BMAPI_VIR_NIC_INFO_EX_VER       10

typedef struct _BM_VIR_NIC_INFO_EX
{
    //  Version is defined as BMAPI_VIR_NIC_INFO_EX_VER.
    //  'version' is required upon input
    U32                 version;

    BM_ADAPTER_INFO     adap_info;

    // Possible value:
    // BMAPI_LM_STATUS_LINK_ACTIVE, BMAPI_LM_STATUS_LINK_DOWN
    U32                 link_status;

    // Driver information
    U8                  driver_name[BMAPI_MAX_DRV_NAME];
    U32                 driver_size;
    U8                  driver_interface_type[BMAPI_MAX_DRV_INT_TYPE];
    U32                 driver_interface_version_major;
    U32                 driver_interface_version_minor;

    // Team name.
    S8                  team_name[BMAPI_MAX_TEAM_NAME_LEN];

    // Team ID
    U32                 team_id;

    // The unit value is Kbps. The actual speed value will be truncated
    // to the lower bound of Kbps. For example, if the speed is 14.4 Kbps,
    // the value will be 14. If the speed is less than 1 Kbps, the value
    // will be 0.
    U32                 line_speed_Kbps;

    // Team Type. BMAPI_TEAM_BRCM_LBFO == Broadcom LBFO,
    // BMAPI_TEAM_802_3_AD == 802.3ad, BMAPI_TEAM_FEC_GEC == FEC/GEC
    // BMAPI_TEAM_SLB_AFD == SLB (Auto Fallback Disabled)
    U32                 team_type;

    // For NDIS devices.
    U32                 mtu;

    // Available for NDIS 6.0 and later.
    // Same as 'duplex_mode' in BM_BRCM_ADAPTER_INFO and BM_LINK_STATUS
    U32                 duplex_mode;

    // Available for NDIS 6.0 and later.
    // Definitions for 'rss_caps' is the same as 'rss_caps' in
    // BM_ADAPTER_INFO_EX.
    U32                 rss_caps;
    U32                 rss_num_interrupt_msg;
    U32                 rss_num_rcv_que;

    U32                 reserved[3];

    // 802.3 offload capabilities
    // These three fields valid ONLY for OS prior to (include) Windows 2003.
    // For NDIS 6.0 and later (Vista and Longhorn), use 'ol_task' and
    // 'ol_conn'.
    BM_OFFLOAD_TCP_IP_CHECKSUM  ol_802_3_tcpip_chksum;
    BM_OFFLOAD_TCP_LARGE_SEND   ol_802_3_lso;
    BM_OFFLOAD_TCP_CONNECTION   ol_802_3_tcp_conn;

    // First IPv6 address found for the NIC.
    // Applications can cast like "( LPSOCKADDR )&ipv6.Address" to API
    // such as WSAAddressToString().
    BM_IP_UNICAST_ADDRESS       ipv6;

    // These two fields valid only for NDIS 6.0 and later (Vista and
    // Longhorn).
    BM_TASK_OFFLOAD             ol_task;
    BM_TCP_CONNECTION_OFFLOAD   ol_conn;

    // Available for NDIS 6.0 and later.
    // The link speed is in bits per second. A value of -1 in this member
    // indicates that the link speed is unknown.
    U64                 max_xmit_link_speed;
    U64                 max_rcv_link_speed;
    U64                 xmit_link_speed;
    U64                 rcv_link_speed;

} BM_VIR_NIC_INFO_EX;



// 'mode' in BM_BRCM_ADAPTER_INFO
#define BMAPI_LM_BUS_MODE_UNKNOWN          0
#define BMAPI_LM_BUS_MODE_PCI              1
#define BMAPI_LM_BUS_MODE_PCI_X            2
#define BMAPI_LM_BUS_MODE_PCI_E            3

// 'clock' in BM_BRCM_ADAPTER_INFO
#define BMAPI_LM_BUS_CLOCK_UNKNOWN         0
#define BMAPI_LM_BUS_CLOCK_33MHZ           1
#define BMAPI_LM_BUS_CLOCK_66MHZ           2
#define BMAPI_LM_BUS_CLOCK_133MHZ          3
#define BMAPI_LM_BUS_CLOCK_100MHZ          4
#define BMAPI_LM_BUS_CLOCK_50MHZ           5

// 'bus_size' in BM_BRCM_ADAPTER_INFO
#define BMAPI_LM_BUS_SIZE_UNKNOWN          0
#define BMAPI_LM_BUS_SIZE_32BIT            1
#define BMAPI_LM_BUS_SIZE_64BIT            2
#define BMAPI_LM_BUS_SIZE_PCI_E_1X         3
#define BMAPI_LM_BUS_SIZE_PCI_E_2X         4
#define BMAPI_LM_BUS_SIZE_PCI_E_4X         5
#define BMAPI_LM_BUS_SIZE_PCI_E_8X         6
#define BMAPI_LM_BUS_SIZE_PCI_E_12X        7
#define BMAPI_LM_BUS_SIZE_PCI_E_16X        8
#define BMAPI_LM_BUS_SIZE_PCI_E_32X        9

// 'duplex_mode' in BM_BRCM_ADAPTER_INFO and BM_LINK_STATUS
#define BMAPI_LM_DUPLEX_MODE_UNKNOWN       0
#define BMAPI_LM_DUPLEX_MODE_HALF          1
#define BMAPI_LM_DUPLEX_MODE_FULL          2

// 'drv_state' in BM_BRCM_ADAPTER_INFO
#define BMAPI_DRV_STATE_NORMAL_MODE        0
#define BMAPI_DRV_STATE_MEDIA_FAIL         1   // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_ADAPTER_CHECK      2   // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_DIAG_MODE          5
#define BMAPI_DRV_STATE_NIC_REMOVED        6   // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_LOW_POWER_MODE     7
#define BMAPI_DRV_STATE_USER_SIMULATE_FAIL  16  // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_POWER_OFF_PENDING   32  // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_POWER_OFF           48  // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_POWER_OFF_FAULT     64  // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_POWER_ON_PENDING    80  // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_POWER_ON            96  // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_POWER_ON_FAULT      112 // NT 4, HotPlug PCI
#define BMAPI_DRV_STATE_RESETTING          0xFFFFFFFF
```

```
typedef struct _BM_BRCM_ADAPTER_INFO
{
    // This structure will host all Broadcom proprietary static data.
    // Ststistic data will be host in separate data structure.

    // PCI information.
    U32 bus_no;
    U32 device_no;
    U32 function_no;
    U32 vendor_id;
    U32 device_id;
    U32 subsystem_vendor_id;
    U32 subsystem_id;
    U32 mem_base_low;
    U32 mem_base_high;

    // Possible value:
    // BMAPI_LM_BUS_MODE_UNKNOWN, BMAPI_LM_BUS_MODE_PCI,
    // BMAPI_LM_BUS_MODE_PCI_X, BMAPI_LM_BUS_MODE_PCI_E
    U32 mode;

    // Possible value:
    // BMAPI_LM_BUS_CLOCK_UNKNOWN, BMAPI_LM_BUS_CLOCK_33MHZ,
    // BMAPI_LM_BUS_CLOCK_66MHZ, BMAPI_LM_BUS_CLOCK_133MHZ,
    // BMAPI_LM_BUS_CLOCK_100MHZ, BMAPI_LM_BUS_CLOCK_50MHZ
    U32 clock;

    // Possible value:
    // BMAPI_LM_BUS_SIZE_UNKNOWN, BMAPI_LM_BUS_SIZE_32BIT,
    // BMAPI_LM_BUS_SIZE_64BIT, BMAPI_LM_BUS_SIZE_PCI_E_1X
    // BMAPI_LM_BUS_SIZE_PCI_E_2X, BMAPI_LM_BUS_SIZE_PCI_E_4X
    // BMAPI_LM_BUS_SIZE_PCI_E_8X, BMAPI_LM_BUS_SIZE_PCI_E_12X
    // BMAPI_LM_BUS_SIZE_PCI_E_16X, BMAPI_LM_BUS_SIZE_PCI_E_32X
    U32 bus_size;

    // Chip information.

    // phy_id and phy_addr are not available for 5706 devices.
    /* OUI: bit 31-10;   Model#: bit 9-4;   Rev# bit 3-0. */
    U32 phy_id;

    U32 phy_addr;

    U32 irq;

    // For 570x devices: ChipId:4, ChipRev:4, MetalRev:8
    // For example :
    // 5700 A1 : 0x7000 : chip ID 0x7, chip rev 0x0, metal rev 0x01
    // 5700 B2 : 0x7000 : chip ID 0x7, chip rev 0x1, metal rev 0x02
    // 5701 A3 : 0x0003 : chip ID 0x0, chip rev 0x0, metal rev 0x03
    // 5703 A0 : 0x1000 : chip ID 0x1, chip rev 0x0, metal rev 0x00
    // 5704 A0 : 0x2003 : chip ID 0x2, chip rev 0x0, metal rev 0x00
    //
    // For 5706 devices: ChipId:16, ChipRev:4, MetalRev:8
    // For example :
    // 5706 A0 : 0x5706000 : chip ID 0x5706, chip rev 0x0, metal rev 0x00
    U32 chip_rev;

    U8  mac_address[8];         // Adapter's MAC address.

    // Driver information.
    // not available for 5706 devices.
    U32 mapped_mem_base;

    // Possible value:
    // BMAPI_LM_MEDIA_TYPE_UNKNOWN, BMAPI_LM_MEDIA_TYPE_AUTO,
    // BMAPI_LM_MEDIA_TYPE_UTP, BMAPI_LM_MEDIA_TYPE_BNC,
    // BMAPI_LM_MEDIA_TYPE_AUI, BMAPI_LM_MEDIA_TYPE_FIBER
    U32 media_type;

    // Possible value:
    // BMAPI_LM_LINE_SPEED_UNKNOWN, BMAPI_LM_LINE_SPEED_10MBPS,
    // BMAPI_LM_LINE_SPEED_100MBPS, BMAPI_LM_LINE_SPEED_1000MBPS
    U32 line_speed;

    // Possible value:
```

```
    // BMAPI_LM_DUPLEX_MODE_UNKNOWN, BMAPI_LM_DUPLEX_MODE_HALF
    // BMAPI_LM_DUPLEX_MODE_FULL
    U32 duplex_mode;

    // Possible value:
    // BMAPI_LM_STATUS_LINK_ACTIVE, BMAPI_LM_STATUS_LINK_DOWN
    U32 link_status;

    U8  node_address[8];        // Current network address.
    U32 drv_major_ver;
    U32 drv_minor_ver;

    // Possible value:
    // BMAPI_DRV_STATE_NORMAL_MODE, BMAPI_DRV_STATE_DIAG_MODE,
    // BMAPI_DRV_STATE_NIC_REMOVED, BMAPI_DRV_STATE_LOW_POWER_MODE
    U32 drv_state;

} BM_BRCM_ADAPTER_INFO;



// 'RequestedMediaType' in BM_BRCM_ADAPTER_INFO_EX
// 'requested_media_type' in BM_LINK_STATUS
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_AUTO                      0
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_10MBPS                3
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_10MBPS_FULL_DUPLEX    4
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_100MBPS               5
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_100MBPS_FULL_DUPLEX   6

// Do NOT force speed on 1000Mbps for copper.
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_1000MBPS              7
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_1000MBPS_FULL_DUPLEX  8

#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_1000MBPS                11
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_1000MBPS_FULL_DUPLEX    12
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_2500MBPS                13
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_2500MBPS_FULL_DUPLEX    14
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_AUTONEG_1G_FALLBACK     15
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_AUTONEG_1G_FALLBACK_FULL_DUPLEX     16
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_AUTONEG_2_5G_FALLBACK               17
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_AUTONEG_2_5G_FALLBACK_FULL_DUPLEX   18
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_HARDWARE_DEFAULT                    19
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_10GBPS                  20
#define BMAPI_LM_REQUESTED_MEDIA_TYPE_FIBER_10GBPS_FULL_DUPLEX      21

// 'flow_ctrl' in BM_BRCM_ADAPTER_INFO_EX
// For both NetXtreme and NetXtreme II products,
// following definitions are valid.
#define BMAPI_FLOW_CONTROL_NONE             0x00
#define BMAPI_FLOW_CONTROL_RX_PAUSE         0x01
#define BMAPI_FLOW_CONTROL_TX_PAUSE         0x02

// For NetXtreme II products ***ONLY***
/*
 * This value can be or-ed with RECEIVE_PAUSE and TRANSMIT_PAUSE.  If the
 * auto-negotiation is disabled and the RECEIVE_PAUSE and TRANSMIT_PAUSE
 * bits are set, then flow control is enabled regardless of link partner's
 * flow control capability.  Otherwise, if this bit is set, then flow
 * is negotiated with the link partner.  Values 0x80000000 and 0x80000003 are
 * equivalent.
 */
#define BMAPI_FLOW_CONTROL_AUTO_PAUSE       0x80000000

// 'pcie_speed' in BM_BRCM_ADAPTER_INFO_EX
#define BMAPI_PCIE_SPEED_UNKNOWN            0
#define BMAPI_PCIE_SPEED_2_5_G              25
#define BMAPI_PCIE_SPEED_5_G                50

// 'mode' in BM_BRCM_ADAPTER_INFO
typedef struct _BM_INTERRUPT_INFO
{

    U32                 IntType;
#define BMAPI_NIC_INTR_TYPE_LINE_BASED  0
#define BMAPI_NIC_INTR_TYPE_MSI         1
```

```
    union
    {
        struct
        {
            U8          IntPin;
            U8          IntLine;
        } LineBased;

        struct
        {
            U8          MsiVersion;
#define BMAPI_NIC_INTR_MSI_1_0          1
#define BMAPI_NIC_INTR_MSI_X            2

            U8          Reserved;

            ////////////////////////////////////
            // Availavble only for NX1 devices.
            U16         MsgData;
            U32         MsgAddressLow;
            U32         MsgAddressHigh;
            ////////////////////////////////////

            U32         NumOfEnabledMsgs;
            U32         NumOfCapableMsgs;
        } Msi;
    } Charactristic;
} BM_INTERRUPT_INFO;

#define BMAPI_BRCM_ADAPTER_INFO_EX_VER  13

typedef struct _BM_BRCM_ADAPTER_INFO_EX
{
    // This structure will host all Broadcom proprietary static data.
    // Ststistic data will be host in separate data structure.

    //  Latest version is defined as BMAPI_BRCM_ADAPTER_INFO_EX_VER.
    //  'version' is required upon input
    U32                     version;

    BM_BRCM_ADAPTER_INFO    brcm_info;

    // Defined flags:
#define BMAPI_DRV_RSS_ENABLED       0x00000004  // 1 if driver is doing RSS, 0 if not
#define BMAPI_DRV_ISCSI_BOOT_INTF   0x00000008  // 1 if the interface is an ISCSI boot
interface
#define BMAPI_REMOTE_PHY_CAPABLE    0x00000010  // 1 if the interface capable of remote
PHY
#define BMAPI_REMOTE_PHY_ACTIVE     0x00000020  // 1 if the interface is using remote PHY
// Determine which MFW is currently ruinning in NX2
#define BMAPI_MFW_RUN_UNKNOWN       0x00000000
#define BMAPI_MFW_RUN_IPMI          0x00000040
#define BMAPI_MFW_RUN_UMP           0x00000080
#define BMAPI_MFW_RUN_NCSI          0x000000c0
#define BMAPI_MFW_RUN_NONE          0x000001c0
#define BMAPI_MFW_RUN_MASK          0x000001c0
#define BMAPI_REMOTE_PHY_MODULE_PRESENT 0x00000200  // 1 if the remote PHY module is
present
#define BMAPI_DRV_ISCSI_BOOT_HBA    0x00000400  // 1 if the interface ISCSI boot from HBA,
applicable only for BRCM iSCSI boot
#define BMAPI_DRV_ISCSI_BOOT_NDIS   0x00000800  // 1 if the interface ISCSI boot from
NDIS, applicable only for BRCM iSCSI boot
#define BMAPI_EXT_PHY               0x00001000  // 1 if the port is using external PHY,
applicable only for 57710 family
#define BMAPI_WOL_NO_VAUX           0x00002000  // 1 if the port does not have Vaux power
to support WOL, applicable only for 5706 family
#define BMAPI_DRV_FCOE_BOOT_HBA     0x00004000  // 1 if the interface ISCSI boot from HBA,
applicable only for BRCM FCoE boot
#define BMAPI_VBD_NO_L4             0x40000000  // 1 if it is VBD but no L4 support.
#define BMAPI_EXT_INFO_VALID        0x80000000  // 1 if 'intr_info' is valid
    U32                     SupportedFlags;

    // Not supported by 57710 based NICs
    U32                     RequestedMediaType;
    U32                     DisableAutoNeg;
    // This data apply to dual port BMAPI_BRCM5700 NIC only.
    U32                     DualMACCtrlReg;

    // Max. speed for the NIC.
    // Value is the same as 'line_speed'.
    U32                     MaxSpeed;

    // The unit value is Kbps. The actual speed value will be truncated
    // to the lower bound of Kbps. For example, if the speed is 14.4 Kbps,
    // the value will be 14. If the speed is less than 1 Kbps, the value
    // will be 0.
    U32                     line_speed_Kbps;

    // This field will only be non-zero if the PHY negociated flow control
    // with the far end.
    U32                     flow_ctrl;

    // Not applicable for 4401 NICs.
    U32                     bond_id;

    // PCI-E <-> PCI-X bridge information
    // Currently, only 5708 based devices support this field.
    // Could be: BMAPI_LM_BUS_SIZE_PCI_E_1X, BMAPI_LM_BUS_SIZE_PCI_E_2X,
    // BMAPI_LM_BUS_SIZE_PCI_E_4X, BMAPI_LM_BUS_SIZE_PCI_E_8X,
    // BMAPI_LM_BUS_SIZE_PCI_E_12X, BMAPI_LM_BUS_SIZE_PCI_E_16X,
    // BMAPI_LM_BUS_SIZE_PCI_E_32X
    U32                     pcie_width;

    // Could be: BMAPI_PCIE_SPEED_2_5_G
    U32                     pcie_speed;

    // The field is valid only when 'nic_type' is BMAPI_BRCM5706 and
    // vbd driver is v2.8.2 or later.
    // Bit 0 indicates buffered flash (set to '1') (VBD 2.8.3 and later).
    U32                     nvram_size;

    // 'intr_info' is valid only when 'BMAPI_INTR_INFO_VALID' is set
    // in SupportedFlags.
    // 'intr_info' is currently available only when 'BMAPI_EXT_INFO_VALID'
    // is set.
    BM_INTERRUPT_INFO       intr_info;

    // For BMAPI_BRCM5706 and BMAPI_BRCM57710 only.
    // 0 for primary port, 1 for secondary port, etc...
    U32                     port_id;

    // iSCSI mac address
    // Available only for NX2 devices.
    U8                      iscsi_mac_addr[8];

    // For 57711 families only.
    // 1 if multi function mode (can be set even if function_per_port==1)
    U32                     mf_mode;

    // For 57711 families only.
    // Can be 1, 2 or 4.
    U32                     function_per_port;

    // In virtual function, the boolean value can be used to determine
    // whether the physical link is up or not.
    U32                     physical_link_up;

    // Transceiver data. Available only for BMAPI_BRCM57710 with EVBD driver.
    U8                      vendor_name[16];
    U8                      model_num[16];
    U8                      serial_num[16];
    U8                      revision_num[4];
    U8                      mfg_date[6];

} BM_BRCM_ADAPTER_INFO_EX;



#define BMAPI_MAXIMUM_MEMBERS_PHY_TEAM  8
#define BMAPI_MAXIMUM_MEMBERS_VIR_TEAM  64
```

```
typedef struct _BM_TEAM_INFO
{
    // Team name.
    S8  team_name[BMAPI_MAX_TEAM_NAME_LEN];

    // Team ID
    U32 team_id;

    // Team Type. BMAPI_TEAM_BRCM_LBFO == Broadcom LBFO,
    // BMAPI_TEAM_802_3_AD == 802.3ad, BMAPI_TEAM_FEC_GEC == FEC/GEC
    // BMAPI_TEAM_SLB_AFD == SLB (Auto Fallback Disabled)
    U32 team_type;

    // NetworkAddress. It is set if team_type == BMAPI_TEAM_FEC_GEC.
    S8  network_addr[BMAPI_MAX_MAC_ADDR_LEN];

    // number of valid physical adapters in the team
    U32 phy_count;

    // information of each team member
    BM_ADAPTER_INFO phy_member[BMAPI_MAXIMUM_MEMBERS_PHY_TEAM];

    // number of valid virtual adapters in the team
    U32 vir_count;

    // information for team configuration
    BM_ADAPTER_INFO vir_member[BMAPI_MAXIMUM_MEMBERS_VIR_TEAM];

} BM_TEAM_INFO;



// 'mode' in BM_TEAM_INFO2
#define BMAPI_TEAM_MODE_PRIMARY                 0
#define BMAPI_TEAM_MODE_STAND_BY                1

#define BMAPI_TEAM_MEMBER_STATE_UP              0x00000001  // link up if set (link down
if not set)
#define BMAPI_TEAM_MEMBER_STATE_DISABLED        0x00000002  // adapter has been disabled
#define BMAPI_TEAM_MEMBER_STATE_JOIN_TRAFFIC    0x00000004  // adapter is being used to
balance the network traffic
#define BMAPI_TEAM_MEMBER_STATE_TRUE_PRIMARY    0x00000008  // adapter is a true primary
#define BMAPI_TEAM_MEMBER_STATE_PROMOTED        0x00000010  // adapter is a currently a
promoted primary

typedef struct _BM_TEAM_PHY_INFO
{
    BM_ADAPTER_INFO     nic;

    // The value is set ***ONLY*** via BmapiGetTeamSnapShot2().
    // state information defined as BMAPI_TEAM_MEMBER_STATE_???
    // 'state' in BM_ADAPTER_INFO will be IGNORED.
    U32                 state;

    U32                 reserved[7];

} BM_TEAM_PHY_INFO;

typedef struct _BM_TEAM_VIR_INFO
{
    BM_ADAPTER_INFO     nic;

    // The value is set ***ONLY*** via BmapiGetTeamSnapShot2().
    // state information defined as BMAPI_TEAM_MEMBER_STATE_???
    // 'state' in BM_ADAPTER_INFO will be IGNORED.
    // can be link up, link down and disable
    U32                 state;

    U32                 reserved[8];

} BM_TEAM_VIR_INFO;


#define BMAPI_AF_INET       2
#define BMAPI_AF_INET6      23
```

```
typedef struct _BM_IPADDR
{
    // Currently only BMAPI_AF_INET is supported.
    U16     sin_family;

    union
    {
        // Unused bytes MUST be set to zero.
        U32     addr_ipv4;      // can be cast to in_addr

        U8      addr_ipv6[16];

        // Application can use this field as protocal independent
        // variable to achieve better compatibility between IPv4 and
        // IPv6.
        U8      addr[16];
    };

} BM_IPADDR;


#define BMAPI_TEAM_INFO2_VER            4

typedef struct _BM_TEAM_INFO2
{
    // version, define as BMAPI_TEAM_INFO2_VER
    U32                 version;

    // Team name.
    S8                  team_name[BMAPI_MAX_TEAM_NAME_LEN];

    // Team ID
    U32                 team_id;

    // Team Type. BMAPI_TEAM_BRCM_LBFO == Broadcom LBFO,
    // BMAPI_TEAM_802_3_AD == 802.3ad, BMAPI_TEAM_FEC_GEC == FEC/GEC
    // BMAPI_TEAM_SLB_AFD == SLB (Auto Fallback Disabled)
    U32                 team_type;

    // NetworkAddress. It is set if team_type == BMAPI_TEAM_FEC_GEC.
    S8                  network_addr[BMAPI_MAX_MAC_ADDR_LEN];

    // The value is set ***ONLY*** via BmapiGetTeamSnapShot2().
    // mode of this team, BMAPI_TEAM_MODE_PRIMARY or BMAPI_TEAM_MODE_STAND_BY
    // For team type other than BMAPI_TEAM_SLB_AFD, this should always be
    // BMAPI_TEAM_MODE_PRIMARY.
    U32                 mode;

    // Advanced failover is disable or not. True or false value.
    // True to disable the feature.
    // Advanced failover can be enabled only when all members are Broadcom
    // NICs.
    U32                 disable_adfo;

    // Probe Packet is enabled or not. True or false value.
    // Probe packet can be enabled only for BMAPI_TEAM_BRCM_LBFO and
    // BMAPI_TEAM_SLB_AFD.
    U32                 probe_enable;

    // The frequency in milliseconds that a probe packet is to be sent.
    // Valid only when 'probe_enable' is enabled.
    U32                 probe_freq;

    // The maximum number of retries before failing a team member.
    // Valid only when 'probe_enable' is enabled.
    U32                 probe_retries;

    // The frequency (in milliseconds) a probe packet is to be sent after
    // a dropped probe packet is detected.
    // Valid only when 'probe_enable' is enabled.
    U32                 probe_retry_freq;

    // This is for BMAPI internal use only.
    U64                 context;

    U32                 reserved;
    // number of valid physical adapters in the team
    U32                 phy_count;

    // information of each team member
    BM_TEAM_PHY_INFO    phy_member[BMAPI_MAXIMUM_MEMBERS_PHY_TEAM];

    // number of valid virtual adapters in the team
    U32                 vir_count;

    // information for team configuration
    BM_TEAM_VIR_INFO    vir_member[BMAPI_MAXIMUM_MEMBERS_VIR_TEAM];

    // This is the probe target IP.
    // Valid only when 'probe_enable' is enabled.
    // If IP is all '0', the value will be deleted from configuration.
    // At leaset one probe target should be configured.
    // The application MUST configure probe target ip as the order from
    // 1 to 4.
    BM_IPADDR           probe_target_ip_1;
    BM_IPADDR           probe_target_ip_2;
    BM_IPADDR           probe_target_ip_3;
    BM_IPADDR           probe_target_ip_4;

    // This the IPv4 for the physical NICs. It is one to one matching.
    // Valid only when 'probe_enable' is enabled.
    // All the IP addresses should be checked for uniqueness (there can
    // not be duplicates for all teams). They also should be different
    // than the IP addresses assigned to the NIC by the OS.
    BM_IPADDR           probe_phy_ip[BMAPI_MAXIMUM_MEMBERS_PHY_TEAM];

    // Vlan ID (0-4094) for LiveLink. It is a per team configuration.
    U32                 ll_vlan_id;

    // This the IPv6 for the physical NICs. It is one to one matching.
    // Valid only when 'probe_enable' is enabled.
    // All the IP addresses should be checked for uniqueness (there can
    // not be duplicates for all teams). They also should be different
    // than the IP addresses assigned to the NIC by the OS.
    BM_IPADDR           probe_phy_ipv6[BMAPI_MAXIMUM_MEMBERS_PHY_TEAM];

} BM_TEAM_INFO2;



typedef struct _BM_PHY_NIC_STATISTICS
{
    // Application use this handle number to know which adapter
    // this statistic structure is referring to.
    U32 handle;

    // number of send request completed by miniport
    U32 send_packet;

    // number of send request discarded
    U32 send_discarded;

    // number of send request dispatched to miniport
    U32 send_dispatched;

    // number of receive packets that are indicated to upper layer
    U32 recv_packet;

    // number of receive packets discarded
    U32 recv_discarded;

    // number of receive discarded because it's too big
    U32 recv_oversize;

} BM_PHY_NIC_STATISTICS;



typedef struct _BM_VIR_NIC_STATISTICS
{
    // Application use this handle number to know which adapter
    // this statistic structure is referring to.
    U32 handle;

    // number of send request discarded for various reasons
    U32 send_all_down;

    // number of send request being queued because out of NDIS_PACKET
    U32 send_queued;

    // number of send request completed
    U32 send_completed;

    // number of receive indicated to upper layer successfully
    U32 recv_indicated;

} BM_VIR_NIC_STATISTICS;



typedef struct _BM_TEAM_STATISTICS
{
    // Team name.
    S8                      team_name[BMAPI_MAX_TEAM_NAME_LEN];

    // number of send packets that are forwarded to the team members,
    // derived from physical adapter statistics
    U32                     send_packet;

    // number of send packets that are discarded,
    // derived from physical adapter statistics
    U32                     send_discarded;

    // number of send packets that are discarded because all the team
    // members are malfunctioned, derived from virtual adapter statistics.
    U32                     send_all_down;

    // number of send packets queued by BLF driver waiting for resource
    // derived from virtual adapter statistics.
    U32                     send_queued;

    // number of send packets originated from BLF driver
    U32                     send_adjusted;

    // number of receive packets that are indicated to upper layer,
    // derived from physical adapter statistics
    U32                     recv_packet;

    // number of receive packets discarded,
    // derived from physical adapter statistics
    U32                     recv_discarded;

    // number of receive discarded because it's too big,
    // derived from physical adapter statistics
    U32                     recv_oversize;

    // number of receive indicated to upper layer successfully
    // derived from virtual adapter statistics
    U32                     recv_indicated;

    // physical adapter statistics
    // number of valid physical adapters
    U32                     phy_count;

    // statistics of each team member
    BM_PHY_NIC_STATISTICS   phy_member[BMAPI_MAXIMUM_MEMBERS_PHY_TEAM];

    // virtual adapter statistics
    // number of valid virtual adapters
    U32                     vir_count;

    // statistics of each virtual adapter
    BM_VIR_NIC_STATISTICS   vir_member[BMAPI_MAXIMUM_MEMBERS_VIR_TEAM];

} BM_TEAM_STATISTICS;
```

```
typedef struct
{
    // Application use this handle number to know which adapter
    // this statistic structure is referring to.
    U32 handle;

    // number of send request completed by miniport
    U32 send_packet;

    // number of send request discarded
    U32 send_discarded;

    // number of send request dispatched to miniport
    U32 send_dispatched;

    // number of receive packets that are indicated to upper layer
    U32 recv_packet;

    // number of receive packets discarded
    U32 recv_discarded;

    // number of receive discarded because it's too big
    U32 recv_oversize;

    // Number of probe packets sent for the adapter -
    // Expressed as a quantity.
    U32 probe_sent;

    // The number of retry packets sent for the adapters -
    // Expressed as a quantity.
    U32 probe_retried;

    // The number of time this NIC was activated by LiveLink
    U32 llNicUpCount;

    // The number of time this NIC was de-activated by LiveLink
    U32 llNicDownCount;

    // The number of time this NIC was activated (does not include LiveLink count)
    U32 nicUpCount;

    // The number of time this NIC was de-activated (does not include LiveLink count)
    U32 nicDownCount;

    // reserved for future use
    U32 reserved[60];

} BM_PHY_NIC_STATISTICS_EX;


typedef struct
{
    // Application use this handle number to know which adapter
    // this statistic structure is referring to.
    U32 handle;

    // number of send request discarded for various reasons
    U32 send_all_down;

    // number of send request being queued because out of NDIS_PACKET
    U32 send_queued;

    // number of send request completed
    U32 send_completed;

    // number of receive indicated to upper layer successfully
    U32 recv_indicated;

    // reserved for future use
    U32 reserved[32];

} BM_VIR_NIC_STATISTICS_EX;
```

```
#define BMAPI_TEAM_STATISTICS_EX_VER    2

typedef struct
{
    //  Version is defined as BMAPI_TEAM_STATISTICS_EX_VER.
    //  'version' is required upon input
    U32                     version;

    // Team name.
    S8                      team_name[BMAPI_MAX_TEAM_NAME_LEN];

    // number of send packets that are forwarded to the team members,
    // derived from physical adapter statistics
    U32                     send_packet;

    // number of send packets that are discarded,
    // derived from physical adapter statistics
    U32                     send_discarded;

    // number of send packets that are discarded because all the team
    // members are malfunctioned, derived from virtual adapter statistics.
    U32                     send_all_down;

    // number of send packets queued by BLF driver waiting for resource
    // derived from virtual adapter statistics.
    U32                     send_queued;

    // number of send packets originated from BLF driver
    U32                     send_adjusted;

    // number of receive packets that are indicated to upper layer,
    // derived from physical adapter statistics
    U32                     recv_packet;

    // number of receive packets discarded,
    // derived from physical adapter statistics
    U32                     recv_discarded;

    // number of receive discarded because it's too big,
    // derived from physical adapter statistics
    U32                     recv_oversize;

    // number of receive indicated to upper layer successfully
    // derived from virtual adapter statistics
    U32                     recv_indicated;

    // physical adapter statistics
    // number of valid physical adapters
    U32                     phy_count;

    // statistics of each team member
    BM_PHY_NIC_STATISTICS_EX    phy_member[BMAPI_MAXIMUM_MEMBERS_PHY_TEAM];

    // virtual adapter statistics
    // number of valid virtual adapters
    U32                     vir_count;

    // statistics of each virtual adapter
    BM_VIR_NIC_STATISTICS_EX    vir_member[BMAPI_MAXIMUM_MEMBERS_VIR_TEAM];

    // Number of probe packets sent across all adapters for the team -
    // Expressed as a quantity.
    U32                     probe_sent;
    // The number of retry packets sent across all adapters for the team -
    // Expressed as a quantity.
    U32                     probe_retried;

} BM_TEAM_STATISTICS_EX;



typedef struct _BM_GENERAL_STATISTICS
{
    U32 xmit_ok;                // OID_GEN_XMIT_OK
    U32 rcv_ok;                 // OID_GEN_RCV_OK
    U32 xmit_error;             // OID_GEN_XMIT_ERROR
    U32 rcv_error;              // OID_GEN_RCV_ERROR
    U32 rcv_no_buffer;          // OID_GEN_RCV_NO_BUFFER
    U64 directed_bytes_xmit;    // OID_GEN_DIRECTED_BYTES_XMIT
    U32 directed_frames_xmit;   // OID_GEN_DIRECTED_FRAMES_XMIT
    U64 multicast_bytes_xmit;   // OID_GEN_MULTICAST_BYTES_XMIT
    U32 multicast_frames_xmit;  // OID_GEN_MULTICAST_FRAMES_XMIT
    U64 broadcast_bytes_xmit;   // OID_GEN_BROADCAST_BYTES_XMIT
    U32 broadcast_frames_xmit;  // OID_GEN_BROADCAST_FRAMES_XMIT
    U64 directed_bytes_rcv;     // OID_GEN_DIRECTED_BYTES_RCV
    U32 directed_frames_rcv;    // OID_GEN_DIRECTED_FRAMES_RCV
    U64 multicast_bytes_rcv;    // OID_GEN_MULTICAST_BYTES_RCV
    U32 multicast_frames_rcv;   // OID_GEN_MULTICAST_FRAMES_RCV
    U64 broadcast_bytes_rcv;    // OID_GEN_BROADCAST_BYTES_RCV
    U32 broadcast_frames_rcv;   // OID_GEN_BROADCAST_FRAMES_RCV
    U32 rcv_crc_error;          // OID_GEN_RCV_CRC_ERROR
    U32 transimit_queue_length; // OID_GEN_TRANSMIT_QUEUE_LENGTH
} BM_GENERAL_STATISTICS;


typedef struct _BM_GENERAL_STATISTICS64
{
    U64 xmit_ok;                // OID_GEN_XMIT_OK
    U64 rcv_ok;                 // OID_GEN_RCV_OK
    U64 xmit_error;             // OID_GEN_XMIT_ERROR
    U64 rcv_error;              // OID_GEN_RCV_ERROR
    U64 rcv_no_buffer;          // OID_GEN_RCV_NO_BUFFER
    U64 directed_bytes_xmit;    // OID_GEN_DIRECTED_BYTES_XMIT
    U64 directed_frames_xmit;   // OID_GEN_DIRECTED_FRAMES_XMIT
    U64 multicast_bytes_xmit;   // OID_GEN_MULTICAST_BYTES_XMIT
    U64 multicast_frames_xmit;  // OID_GEN_MULTICAST_FRAMES_XMIT
    U64 broadcast_bytes_xmit;   // OID_GEN_BROADCAST_BYTES_XMIT
    U64 broadcast_frames_xmit;  // OID_GEN_BROADCAST_FRAMES_XMIT
    U64 directed_bytes_rcv;     // OID_GEN_DIRECTED_BYTES_RCV
    U64 directed_frames_rcv;    // OID_GEN_DIRECTED_FRAMES_RCV
    U64 multicast_bytes_rcv;    // OID_GEN_MULTICAST_BYTES_RCV
    U64 multicast_frames_rcv;   // OID_GEN_MULTICAST_FRAMES_RCV
    U64 broadcast_bytes_rcv;    // OID_GEN_BROADCAST_BYTES_RCV
    U64 broadcast_frames_rcv;   // OID_GEN_BROADCAST_FRAMES_RCV
    U64 rcv_crc_error;          // OID_GEN_RCV_CRC_ERROR
    U64 transimit_queue_length; // OID_GEN_TRANSMIT_QUEUE_LENGTH
} BM_GENERAL_STATISTICS64;


#define BM_GENERAL_STATISTICS_EX_VER    1

typedef struct _BM_GENERAL_STATISTICS_EX
{
    //  Version is defined as BM_GENERAL_STATISTICS_EX_VER.
    //  'version' is required upon input
    U32 version;

    U64 SupportedStatistics;
    // The value of 'SupportedStatistics' is the bitwise OR of the following
    // flags.
// The data in the 'xmit_ok' member is valid.
#define BMAPI_STAT_GEN_XMIT_OK                              0x00000001
// The data in the 'rcv_ok' member is valid.
#define BMAPI_STAT_GEN_RCV_OK                               0x00000002
// The data in the 'xmit_error' member is valid.
#define BMAPI_STAT_GEN_XMIT_ERROR                           0x00000004
// The data in the 'rcv_error' member is valid.
#define BMAPI_STAT_GEN_RCV_ERROR                            0x00000008
// The data in the 'rcv_no_buffer' member is valid.
#define BMAPI_STAT_GEN_RCV_NO_BUFFER                        0x00000010
// The data in the 'directed_bytes_xmit' member is valid.
#define BMAPI_STAT_GEN_DIRECTED_BYTES_XMIT                  0x00000020
// The data in the 'directed_frames_xmit' member is valid.
#define BMAPI_STAT_GEN_DIRECTED_FRAMES_XMIT                 0x00000040
// The data in the 'multicast_bytes_xmit' member is valid.
#define BMAPI_STAT_GEN_MULTICAST_BYTES_XMIT                 0x00000080
// The data in the 'multicast_frames_xmit' member is valid.
#define BMAPI_STAT_GEN_MULTICAST_FRAMES_XMIT                0x00000100
// The data in the 'broadcast_bytes_xmit' member is valid.
```

```
#define BMAPI_STAT_GEN_BROADCAST_BYTES_XMIT                 0x00000200
// The data in the 'broadcast_frames_xmit' member is valid.
#define BMAPI_STAT_GEN_BROADCAST_FRAMES_XMIT                0x00000400
// The data in the 'directed_bytes_rcv' member is valid.
#define BMAPI_STAT_GEN_DIRECTED_BYTES_RCV                   0x00000800
// The data in the 'directed_frames_rcv' member is valid.
#define BMAPI_STAT_GEN_DIRECTED_FRAMES_RCV                  0x00001000
// The data in the 'multicast_bytes_rcv' member is valid.
#define BMAPI_STAT_GEN_MULTICAST_BYTES_RCV                  0x00002000
// The data in the 'multicast_frames_rcv' member is valid.
#define BMAPI_STAT_GEN_MULTICAST_FRAMES_RCV                 0x00004000
// The data in the 'broadcast_bytes_rcv' member is valid.
#define BMAPI_STAT_GEN_BROADCAST_BYTES_RCV                  0x00008000
// The data in the 'broadcast_frames_rcv' member is valid.
#define BMAPI_STAT_GEN_BROADCAST_FRAMES_RCV                 0x00010000
// The data in the 'rcv_crc_error' member is valid.
#define BMAPI_STAT_GEN_RCV_CRC_ERROR                        0x00020000
// The data in the 'transimit_queue_length' member is valid.
#define BMAPI_STAT_GEN_TRANSMIT_QUEUE_LENGTH                0x00040000
// The data in the 'rcv_discards' member is valid.
#define BMAPI_STAT_GEN_RCV_DISCARDS                         0x00080000
// The data in the 'bytes_rcv' member is valid.
#define BMAPI_STAT_GEN_BYTES_RCV                            0x00100000
// The data in the 'bytes_xmit' member is valid.
#define BMAPI_STAT_GEN_BYTES_XMIT                           0x00200000
// The data in the 'xmit_discards' member is valid.
#define BMAPI_STAT_GEN_XMIT_DISCARDS                        0x00400000

    U64 xmit_ok;                // OID_GEN_XMIT_OK
    U64 rcv_ok;                 // OID_GEN_RCV_OK
    U64 xmit_error;             // OID_GEN_XMIT_ERROR
    U64 rcv_error;              // OID_GEN_RCV_ERROR
    U64 rcv_no_buffer;          // OID_GEN_RCV_NO_BUFFER
    U64 directed_bytes_xmit;    // OID_GEN_DIRECTED_BYTES_XMIT
    U64 directed_frames_xmit;   // OID_GEN_DIRECTED_FRAMES_XMIT
    U64 multicast_bytes_xmit;   // OID_GEN_MULTICAST_BYTES_XMIT
    U64 multicast_frames_xmit;  // OID_GEN_MULTICAST_FRAMES_XMIT
    U64 broadcast_bytes_xmit;   // OID_GEN_BROADCAST_BYTES_XMIT
    U64 broadcast_frames_xmit;  // OID_GEN_BROADCAST_FRAMES_XMIT
    U64 directed_bytes_rcv;     // OID_GEN_DIRECTED_BYTES_RCV
    U64 directed_frames_rcv;    // OID_GEN_DIRECTED_FRAMES_RCV
    U64 multicast_bytes_rcv;    // OID_GEN_MULTICAST_BYTES_RCV
    U64 multicast_frames_rcv;   // OID_GEN_MULTICAST_FRAMES_RCV
    U64 broadcast_bytes_rcv;    // OID_GEN_BROADCAST_BYTES_RCV
    U64 broadcast_frames_rcv;   // OID_GEN_BROADCAST_FRAMES_RCV
    U64 rcv_crc_error;          // OID_GEN_RCV_CRC_ERROR
    U64 transimit_queue_length; // OID_GEN_TRANSMIT_QUEUE_LENGTH

    ///////////////////////////////////////////////
    // Available only for Vista and later Windows.
    U64 rcv_discards;           // OID_GEN_RCV_DISCARDS
    U64 bytes_rcv;              // OID_GEN_BYTES_RCV
    U64 bytes_xmit;             // OID_GEN_BYTES_XMIT
    U64 xmit_discards;          // OID_GEN_XMIT_DISCARDS
    ///////////////////////////////////////////////

} BM_GENERAL_STATISTICS_EX;



typedef struct _BM_ETHERNET_STATISTICS
{
    U32 rcv_error_alignment;    //OID_802_3_RCV_ERROR_ALIGNMENT
    U32 xmit_one_collision;     //OID_802_3_XMIT_ONE_COLLISION
    U32 xmit_more_collisions;   //OID_802_3_XMIT_MORE_COLLISIONS
    U32 xmit_deferred;          //OID_802_3_XMIT_DEFERRED
    U32 xmit_max_collisions;    //OID_802_3_XMIT_MAX_COLLISIONS
    U32 rcv_overrun;            //OID_802_3_RCV_OVERRUN
    U32 xmit_underrun;          //OID_802_3_XMIT_UNDERRUN
    U32 xmit_heartbeat_failure; //OID_802_3_XMIT_HEARTBEAT_FAILURE
    U32 xmit_times_crs_lost;    //OID_802_3_XMIT_TIMES_CRS_LOST
    U32 xmit_late_collisions;   //OID_802_3_XMIT_LATE_COLLISIONS
} BM_ETHERNET_STATISTICS;
```

```
typedef struct _BM_ETHERNET_STATISTICS64
{
    U64 rcv_error_alignment;    //OID_802_3_RCV_ERROR_ALIGNMENT
    U64 xmit_one_collision;     //OID_802_3_XMIT_ONE_COLLISION
    U64 xmit_more_collisions;   //OID_802_3_XMIT_MORE_COLLISIONS
    U64 xmit_deferred;          //OID_802_3_XMIT_DEFERRED
    U64 xmit_max_collisions;    //OID_802_3_XMIT_MAX_COLLISIONS
    U64 rcv_overrun;            //OID_802_3_RCV_OVERRUN
    U64 xmit_underrun;          //OID_802_3_XMIT_UNDERRUN
    U64 xmit_heartbeat_failure; //OID_802_3_XMIT_HEARTBEAT_FAILURE
    U64 xmit_times_crs_lost;    //OID_802_3_XMIT_TIMES_CRS_LOST
    U64 xmit_late_collisions;   //OID_802_3_XMIT_LATE_COLLISIONS
} BM_ETHERNET_STATISTICS64;



#define BM_ETHERNET_STATISTICS_EX_VER   1

typedef struct _BM_ETHERNET_STATISTICS_EX
{
    //  Version is defined as BM_ETHERNET_STATISTICS_EX_VER.
    //  'version' is required upon input
    U32 version;

    U64 SupportedStatistics;
    // The value of 'SupportedStatistics' is the bitwise OR of the following
    // flags.
// The data in the 'rcv_error_alignment' member is valid.
#define BMAPI_STAT_802_3_RCV_ERROR_ALIGNMENT                0x00000001
// The data in the 'xmit_one_collision' member is valid.
#define BMAPI_STAT_802_3_XMIT_ONE_COLLISION                 0x00000002
// The data in the 'xmit_more_collisions' member is valid.
#define BMAPI_STAT_802_3_XMIT_MORE_COLLISIONS               0x00000004
// The data in the 'xmit_deferred' member is valid.
#define BMAPI_STAT_802_3_XMIT_DEFERRED                      0x00000008
// The data in the 'xmit_max_collisions' member is valid.
#define BMAPI_STAT_802_3_XMIT_MAX_COLLISIONS                0x00000010
// The data in the 'rcv_overrun' member is valid.
#define BMAPI_STAT_802_3_RCV_OVERRUN                        0x00000020
// The data in the 'xmit_underrun' member is valid.
#define BMAPI_STAT_802_3_XMIT_UNDERRUN                      0x00000040
// The data in the 'xmit_heartbeat_failure' member is valid.
#define BMAPI_STAT_802_3_XMIT_HEARTBEAT_FAILUR              0x00000080
// The data in the 'xmit_times_crs_lost' member is valid.
#define BMAPI_STAT_802_3_XMIT_TIMES_CRS_LOST                0x00000100
// The data in the 'xmit_late_collisions' member is valid.
#define BMAPI_STAT_802_3_XMIT_LATE_COLLISIONS               0x00000200

    U64 rcv_error_alignment;    //OID_802_3_RCV_ERROR_ALIGNMENT
    U64 xmit_one_collision;     //OID_802_3_XMIT_ONE_COLLISION
    U64 xmit_more_collisions;   //OID_802_3_XMIT_MORE_COLLISIONS
    U64 xmit_deferred;          //OID_802_3_XMIT_DEFERRED
    U64 xmit_max_collisions;    //OID_802_3_XMIT_MAX_COLLISIONS
    U64 rcv_overrun;            //OID_802_3_RCV_OVERRUN
    U64 xmit_underrun;          //OID_802_3_XMIT_UNDERRUN
    U64 xmit_heartbeat_failure; //OID_802_3_XMIT_HEARTBEAT_FAILURE
    U64 xmit_times_crs_lost;    //OID_802_3_XMIT_TIMES_CRS_LOST
    U64 xmit_late_collisions;   //OID_802_3_XMIT_LATE_COLLISIONS

} BM_ETHERNET_STATISTICS_EX;



typedef struct _BM_BRCM_STATISTICS
{
    // Statistics maintained by Receive MAC.
    U64 ifHCInOctets;
    U64 etherStatsFragments;
    U64 ifHCInUcastPkts;
    U64 ifHCInMulticastPkts;
    U64 ifHCInBroadcastPkts;
    U64 dot3StatsFCSErrors;
    U64 dot3StatsAlignmentErrors;
    U64 xonPauseFramesReceived;
    U64 xoffPauseFramesReceived;
    U64 macControlFramesReceived;
```

```
U64 xoffStateEntered;

//////////////////////////
// Not available for 57710
U64 dot3StatsFramesTooLong;
//////////////////////////

U64 etherStatsJabbers;
U64 etherStatsUndersizePkts;

//////////////////////////
// Not available for 5706
// Not available for 57710
U64 inRangeLengthError;
U64 outRangeLengthError;
//////////////////////////

//////////////////////////
// Not available for 57710
U64 etherStatsPkts64Octets;
U64 etherStatsPkts65Octetsto127Octets;
U64 etherStatsPkts128Octetsto255Octets;
U64 etherStatsPkts256Octetsto511Octets;
U64 etherStatsPkts512Octetsto1023Octets;
U64 etherStatsPkts1024Octetsto1522Octets;
//////////////////////////

//////////////////////////
// Not available for 57710
// Not available for 5706
U64 etherStatsPkts1523Octetsto2047Octets;
U64 etherStatsPkts2048Octetsto4095Octets;
U64 etherStatsPkts4096Octetsto8191Octets;
U64 etherStatsPkts8192Octetsto9022Octets;
//////////////////////////

// Statistics maintained by Transmit MAC.
U64 ifHCOutOctets;
U64 etherStatsCollisions;
U64 outXonSent;
U64 outXoffSent;
U64 flowControlDone;
U64 dot3StatsInternalMacTransmitErrors;

U64 dot3StatsSingleCollisionFrames;
U64 dot3StatsMultipleCollisionFrames;
U64 dot3StatsDeferredTransmissions;
U64 dot3StatsExcessiveCollisions;
U64 dot3StatsLateCollisions;

//////////////////////////
// Not available for 5706
// Not available for 57710
U64 dot3Collided2Times;
U64 dot3Collided3Times;
U64 dot3Collided4Times;
U64 dot3Collided5Times;
U64 dot3Collided6Times;
U64 dot3Collided7Times;
U64 dot3Collided8Times;
U64 dot3Collided9Times;
U64 dot3Collided10Times;
U64 dot3Collided11Times;
U64 dot3Collided12Times;
U64 dot3Collided13Times;
U64 dot3Collided14Times;
U64 dot3Collided15Times;
//////////////////////////

U64 ifHCOutUcastPkts;
U64 ifHCOutMulticastPkts;
U64 ifHCOutBroadcastPkts;
U64 dot3StatsCarrierSenseErrors;

/////////////////////////
// Not available for 5706
```

```
    U64 ifOutDiscards;
    /////////////////////////

    ////////////////////////////////////////////
    // Not available for 5706 monolithic driver
    U64 ifOutErrors;
    ////////////////////////////////////////////

    // Statistics maintained by Receive List Placement.
    //////////////////////////
    // Not available for 5706
    // Not available for 57710
    U64 COSIfHCInPkts[16];
    U64 COSFramesDroppedDueToFilters;
    U64 nicDmaWriteQueueFull;
    U64 nicDmaWriteHighPriQueueFull;
    //////////////////////////

    //////////////////////////
    // Not available for 57710
    U64 nicNoMoreRxBDs;
    //////////////////////////

    //////////////////////////
    // Not available for 5706 VBD
    // Not available for 57710
    U64 ifInDiscards;
    //////////////////////////

    U64 ifInErrors;

    //////////////////////////
    // Not available for 5706
    // Not available for 57710
    U64 nicRecvThresholdHit;
    //////////////////////////

    // Statistics maintained by Send Data Initiator.
    //////////////////////////
    // Not available for 5706
    // Not available for 57710
    U64 COSIfHCOutPkts[16];
    U64 nicDmaReadQueueFull;
    U64 nicDmaReadHighPriQueueFull;
    U64 nicSendDataCompQueueFull;
    //////////////////////////

    // Statistics maintained by Host Coalescing.
    //////////////////////////
    // Not available for 5706
    // Not available for 57710
    U64 nicRingSetSendProdIndex;
    U64 nicRingStatusUpdate;
    U64 nicInterrupts;
    U64 nicAvoidedInterrupts;
    U64 nicSendThresholdHit;
    //////////////////////////

} BM_BRCM_STATISTICS;


#define BM_BRCM_STATISTICS_EX_VER    1

typedef struct _BM_BRCM_STATISTICS_EX
{
    // version must filled on input
    U64                 version;

    BM_BRCM_STATISTICS  brcm_statistics;

    // -1 if driver does not support this counter
    U64                 TxLargeSendFrames;

} BM_BRCM_STATISTICS_EX;
```

```
typedef struct _BM_LINK_STATUS
{
    U32 link_status;            // 1 = Link Pass, 0 = Link Fail
    U32 local_Rx_status;        // 1 = Local Receive OK
                                // 0 = Local Receive Error
    U32 remote_Rx_status;       // 1 = Remote Receive OK
                                // 0 = Remote Receive Error
    U32 auto_negotiation_mode;  // 1 = AutoNegotiation is enabled
                                // 0 = Forced speed
    U32 requested_media_type;   // BMAPI_LM_REQUESTED_MEDIA_TYPE_AUTO
                                // BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_10MBPS
                                // BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_10MBPS_FULL_DUPLEX
                                // BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_100MBPS
                                // BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_100MBPS_FULL_DUPLEX
                                // BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_1000MBPS
                                // BMAPI_LM_REQUESTED_MEDIA_TYPE_UTP_1000MBPS_FULL_DUPLEX
    U32 duplex_mode;            // 0 = half duplex
                                // 1 = full duplex
    U32 link_speed;             // BMAPI_LM_LINE_SPEED_UNKNOWN
                                // BMAPI_LM_LINE_SPEED_10MBPS
                                // BMAPI_LM_LINE_SPEED_100MBPS
                                // BMAPI_LM_LINE_SPEED_1000MBPS

} BM_LINK_STATUS;



#define BMAPI_LINK_STATUS_EX_VER    10

typedef struct
{
    U32                 version;
    BM_LINK_STATUS      link_status;

    // The unit value is Kbps. The actual speed value will be truncated
    // to the lower bound of Kbps. For example, if the speed is 14.4 Kbps,
    // the value will be 14. If the speed is less than 1 Kbps, the value
    // will be 0.
    U32                 line_speed_Kbps;

    // Boolean flag to indicate whether driver is running.
    U32                 driver_loaded;

    // Driver's major and minor version number.
    U32                 major_version_number;
    U32                 minor_version_number;

    // Boolean flag to denote if DHCP is enabled.
    // Not available in non Windows platforms.
    U32                 dhcp_enabled;

    // IP address is "192.168.0.1" format. (REG_MULTI_SZ or REG_SZ)
    S8                  ip_addr[BMAPI_MAX_IP_ADDR_LEN];

    // subnet mask is "255.255.0.0" format. (REG_MULTI_SZ or REG_SZ)
    S8                  subnet_mask[BMAPI_MAX_IP_ADDR_LEN];

    // defaule gateway is "192.168.0.1" format. (REG_MULTI_SZ or REG_SZ)
    S8                  default_gateway[BMAPI_MAX_IP_ADDR_LEN];

    S8                  current_mac_addr[BMAPI_MAX_MAC_ADDR_LEN];
    S8                  permanent_mac_addr[BMAPI_MAX_MAC_ADDR_LEN];

    // First IPv6 address found for the NIC.
    // Applications can cast like "( LPSOCKADDR )&ipv6.Address" to API
    // such as WSAAddressToString().
    BM_IP_UNICAST_ADDRESS       ipv6;

    // 802.3 offload capabilities
    // These three fields valid ONLY for OS prior to (include) Windows 2003.
    // For NDIS 6.0 and later (Vista and Longhorn), use 'ol_task' and
    // 'ol_conn'.
    BM_OFFLOAD_TCP_IP_CHECKSUM  ol_802_3_tcpip_chksum;
    BM_OFFLOAD_TCP_LARGE_SEND   ol_802_3_lso;
    BM_OFFLOAD_TCP_CONNECTION   ol_802_3_tcp_conn;

    // These two fields valid only for NDIS 6.0 and later (Vista and
    // Longhorn).
    BM_TASK_OFFLOAD             ol_task;
    BM_TCP_CONNECTION_OFFLOAD   ol_conn;

    // This is the handle to the NIC.
    // The field is added to support BmapiRetrieveMultiLinkStatus().
    U32                 handle;

    // First IPv6 Gateway address found for the NIC.
    // Only available for Vista or later.
    // Applications can cast like "( LPSOCKADDR )&gateway_v6.Address" to API
    // such as WSAAddressToString().
    BM_IP_ADAPTER_GATEWAY_ADDRESS   gateway_v6;

    // 'ipv6_flags' in BM_ADAPTER_INFO_EX
    U32                 ipv6_flags;

    // return code for this structure.
    // The field is added to support BmapiRetrieveMultiLinkStatus().
    // Applications must check BmapiRetrieveMultiLinkStatus() to be BMAPI_OK
    // before using 'return_code' from each individual structure.
    U32                 return_code;

    // Same 'SupportedFlags' definition as in BM_BRCM_ADAPTER_INFO_EX.
    U32                 SupportedFlags;

    // For VBD and NDIS devices. Mmonolithic driver is not supported.
    U32                 mtu;

} BM_LINK_STATUS_EX;



/****************************************************************************
 *  OBSOLETE starts here
 ****************************************************************************/
typedef struct _BM_BRCM_NIC_PARAM_REQ
{
    U32     num_of_params;
    U8      param_id_List[1];
} BM_BRCM_NIC_PARAM_REQ;



typedef struct _BM_BRCM_NIC_PARAM_DATA
{
    U32     param_id;
    U32     param_type;
    U32     param_data_len;
    U32     result;
    U32     offset_next_param;
    U8      param_data[1];
} BM_BRCM_NIC_PARAM_DATA;
/****************************************************************************
 *  OBSOLETE ends here
 ****************************************************************************/



#define BMAPI_NIC_PCI_INFO_VER          2

typedef struct _BM_NIC_PCI_INFO
{
    //  Version is defined as BMAPI_NIC_PCI_INFO_VER.
    //  'version' is required upon input
    U16     version;

    // If PCI IDs information is not available, the value will be -1 and
    // 'manufacturer' will be "".
    U16     vendor_id;
    U16     device_id;
    U16     subsystem_vendor_id;
    U16     subsystem_id;
    U8      manufacturer[BMAPI_MAX_MFG_LEN];

    // hardware revision
    U32     revision;

    // PCI bus information. If these information are not available,
    // data will be set to -1.
    U32     bus_no;
    U32     device_no;
    U32     function_no;

    // PCI slot number labeled next to motherboard (W2k only).
    // This is an optional field (at least, LOM will not have it).
    // If it is not available, this field will be set to -1.
    U32     ui_number;

    // PCI information
    // Valid for Broadcom NIC only.
    U32     mem_base_low;
    U32     mem_base_high;

    // PCI information
    // No valid if the NIC is on MSI.
    U32     irq;

} BM_NIC_PCI_INFO;



#define BMAPI_VBD_DEV_CONFIG            0x00000001
#define BMAPI_VBD_DEV_PRESENT           0x00000002

#define BMAPI_VBD_ENUM_INFO_VER         4

typedef struct _BM_VBD_ENUM_INFO
{
    //  Version is defined as BMAPI_VBD_ENUM_INFO_VER.
    //  'version' is required upon input
    U32     version;

    // BMAPI_VBD_DEV_CONFIG: it is configured to be enumerated by VBD
    // BMAPI_VBD_DEV_PRESENT: it is currently enumerated by VBD
    //
    // ATTENTION: When using BmapiSetVbdEnumInfo(), BMAPI_VBD_DEV_CONFIG bit
    //            will be used to configure the VBD to enumerate virtual
    //            devices. Other bits will be ignored.
    //
    // NOTE: WSD will be enumerated only when NDIS is enumerated. If only
    //        WSD is configured, BMAPI will fail BmapiSetVbdEnumInfo().
    U32     ndis_flag;
    U32     wsd_flag;
    U32     iscsi_flag;

    // 1. If the virtual device is not enuemrated properly, the handle will
    //    be set to BMAPI_INVALID_NIC_HANDLE.
    // 2. Check 'flags' in BM_ADAPTER_INFO_EX structure. If
    //    BMAPI_NIC_NO_MINOPORT_DRV is set, no driver is installed for the
    //    device.
    // 3. BmapiSetVbdEnumInfo() will ignore these information.
    U32     ndis_handle;
    U32     wsd_handle;
    U32     iscsi_handle;
    U32     diag_handle;

    // These are information for FCOE. The usage for fcoe_flag is the same
    // as iscsi_flag.
    U32     fcoe_flag;
    U32     fcoe_handle;

} BM_VBD_ENUM_INFO;



// ASF definition
#define BMAPI_MAX_ALERTDATA                             8
#define BMAPI_ASF_SPEC_1_0_ALERTDATA_ELEMENT_LENGTH     12
```

```
#define BMAPI_MAX_CONTROLDATA                           8
#define BMAPI_MAX_SPEC_1_0_CONTROLDATA_ELEMENT_LENGTH   4

// since structure ASF_ADDR has extra 2 bytes other than FixedSMBusAddresses,
// to align the structure to LONG, BMAPI_MAX_FIXEDSMBUSADDR should use
// (multiple of 4) - 2. In this case, we are using 14.
#define BMAPI_MAX_FIXEDSMBUSADDR                        14

// speed defines for BM_ASF_MISC
#define BMAPI_ASF_SPEED_10_100                          0
#define BMAPI_ASF_SPEED_10                              1
#define BMAPI_ASF_SPEED_100                             2
#define BMAPI_ASF_SPEED_1000                            3
#define BMAPI_ASF_SPEED_ALL                             4
#define BMAPI_ASF_SPEED_MASK                            0x7

#define BMAPI_ASF_HALF_DUPLEX                           0x20
#define BMAPI_ASF_PAUSE_NOT_CAPABLE                     0x40
#define BMAPI_ASF_AUTO_DISABLE                          0x80

typedef struct _BM_ASF_INFO
{
    U8      MinWatchdogResetValue;
    U8      MinPollingInterval;
    U16     SystemID;

    U8      IANAManufacturerID[4];

    U8      FeatureFlags;       /* added in ASF 2.0 */
    U8      Reserved[3];
} BM_ASF_INFO;

typedef struct _BM_ASF_ALERTDATA
{
    U8      DeviceAddress;
    U8      Command;
    U8      DataMask;
    U8      CompareValue;

    U8      EventSensorType;
    U8      EventType;
    U8      EventOffset;
    U8      EventSourceType;

    U8      EventSeverity;
    U8      SensorNumber;
    U8      Entity;
    U8      EntityInstance;
} BM_ASF_ALERTDATA;

typedef struct _BM_ASF_ALRT
{
    U8                      AssertionEventMask;     /* added in ASF 2.0 */
    U8                      DeassertionEventMask;   /* added in ASF 2.0 */
    U8                      NumberOfAlerts;
    U8                      ArrayElementLength;

    BM_ASF_ALERTDATA        AsfAlertData[BMAPI_MAX_ALERTDATA];
} BM_ASF_ALRT;

typedef struct _BM_ASF_CONTROLDATA
{
    U8      Function;
    U8      DeviceAddress;
    U8      Command;
    U8      DataValue;
} BM_ASF_CONTROLDATA;

typedef struct _BM_ASF_RCTL
{
    U8                      NumberOfControl;
    U8                      ArrayElementLength;
    /* to align to dword */
    U8                      Reserved[2];
    BM_ASF_CONTROLDATA      AsfControlData[BMAPI_MAX_CONTROLDATA];
} BM_ASF_RCTL;
```

```
typedef struct _BM_ASF_RMCP
{
    U8      RemoteControlCapabilities[7];
    U8      RMCPCompletionCode;

    U8      RMCPIANA[4];

    U8      RMCPSpecialCommand;
    U8      RMCPSpecialCommandParameter[2];
    U8      RMCPBootOptions[2];
    U8      RMCPOEMParameters[2];

    /* to align to dword */
    U8      filler;
} BM_ASF_RMCP;

typedef struct _BM_ASF_ADDR
{
    U8      SEEPROMAddress;
    U8      NumberOfDevices;
    U8      FixedSMBusAddresses[BMAPI_MAX_FIXEDSMBUSADDR];
} BM_ASF_ADDR;

struct BM_ASF_CFG
{
    U32     EnableASF           : 1;    // mirror of BM_FW_FEATURE_CONFIG.asf_enable
    U32     EnableHeartBeat     : 1;
    U32     EnableRMCP          : 1;
    U32     EnablePET           : 1;

    // Enable/Disable WoL on ARP or RMCP packet, while in OS absent
    U32     EnableASFWoL        : 1;

    // Enable/Disable our NIC as a passive Slave Mode.
    // No ARP or Scan on the SMBus.
    U32     EnableSMBusScan     : 1;

    U32     EnableSecureRMCP    : 1;    // ASF 2.0 - Secure RMCP
    U32     EnableSecureOnly    : 1;    // No ASF 1.0 Compatibility

    U32     reserved2           : 7;

    // Send OEM specific "ASD Ready" message after initialization
    U32     EnableSendAsdReady  : 1;

    U32     reserved            : 16;
};

typedef struct _BM_ASF_MISC
{
    union {
        struct BM_ASF_CFG   Config;
        U32                 ConfigData;
    } cfg;

    U8      UuidGuid[16];

    U16     HeartBeatTimeValue;     /* in second */
    U8      reserved1[2];

    U8      DelayedPollTimeValue;   /* in second */
    U8      Speed;                  /* see speed defines above */
    U8      ReTransmitTimeValue;    /* in second */
    U8      LegacyPollTimeValue;    /* in second */

    U8      SysIP[4];
    U8      ManagementConsolIP[4];
    U8      GateWayIP[4];
    U8      subnetMask[4];
    U8      communityName[BMAPI_MAX_COMMUNITY_NAME_LEN];
    U8      reserved2[12];
} BM_ASF_MISC;

#define BMAPI_MAX_INIT_DATA 50
```

```
typedef struct _BM_SMB_INIT_DATA
{
    U8      addr;
    U8      index;
    U8      andMask;
    U8      orMask;
} BM_SMB_INIT_DATA;

#define BMAPI_MAX_KEY_SIZE    20        /* 160 bits */

typedef struct
{
    U32     kgsize;
    U8      kg[BMAPI_MAX_KEY_SIZE];
    U32     kosize;
    U8      ko[BMAPI_MAX_KEY_SIZE];
    U32     kasize;
    U8      ka[BMAPI_MAX_KEY_SIZE];

    /* Security Policy */
#define BMAPI_ASF_CAN_POWER_UP          (1<<24)
#define BMAPI_ASF_CAN_RESET             (1<<25)
#define BMAPI_ASF_CAN_RESET_POWER       (1<<26)
#define BMAPI_ASF_CAN_POWER_DOWN        (1<<27)
#define BMAPI_ASF_RIGHTS_MASK           0x0f000000

    U32     op_rights;          /* operator rights bitfield */
    U32     admin_rights;       /* administrator rights bitfield */
    U32     session_timeout;    /* in seconds */
} BM_ASF20_T;

//  We will use BMAPI_ASF_T_VERSION to identify the current version of
//  BM_ASF_TABLE.
//  Whenever the table is changed for release, BMAPI_ASF_T_VERSION should
//  increase.
#define BMAPI_ASF_T_VERSION         8

typedef struct _BM_ASF_TABLE
{
    U8                  version;    // version of the table

/****************************************************************************
 *  Following fields valid only for ASF and IMPI
 *  For IPMI, only 'EnableASF' in BM_ASF_CFG can be set/clear.
 ****************************************************************************/
    U8                  smbusAddr;  // forced NIC smbus address, 0 means
                                    // use ARP, other value to force
                                    // smbus address
    U16                 size;
    BM_ASF_INFO         AsfInfo;
    BM_ASF_ALRT         AsfAlert;
    BM_ASF_RCTL         AsfRCtl;
    BM_ASF_RMCP         AsfRmcp;
    BM_ASF_ADDR         AsfAddr;
    BM_ASF_MISC         AsfGui;
    BM_SMB_INIT_DATA    smbInitData[BMAPI_MAX_INIT_DATA];

    BM_ASF20_T          Asf20;      // Added in version 6 for ASF 2.0,
                                    // ASF 1.0 firware will ignore this field.

    U8                  reserve[1024];

    U32                 chksum;

/****************************************************************************
 *  ASF and IMPI only fields end here.
 ****************************************************************************/

    // Management firmware version description string.
    U8                  firmware_ver_desc[BMAPI_FW_MAX_DESCRIPTION_LEN];

    U32                 fw_asfcfg_ver;      /* firmware asfcfg version */
    U32                 fw_asfcfg_size;     /* firmware asfcfg size */

#define  BM_ASF_2_0_SUPPORT         0x00000001   //  ASF  Management  Firmware  with  ASF  2.0
support
```

```
#define BM_IPMI_SUPPORT         0x00000002  // IPMI Management Firmware
#define BM_UMP_SUPPORT          0x00000004  // UMP Management Firmware
    U32                 flags;

} BM_ASF_TABLE;



#define BM_ASF_MBOX_STATUS_VER      1

typedef struct _BM_ASF_MBOX_STATUS
{
    //  Latest version is defined as BM_ASF_MBOX_STATUS_VER.
    //  'version' is required when calling BMAPI APIs
    U32     version;

    // 'length' field will be ignored in BmapiSetASFMailboxStatus().
    U32     length;             /* mailbox size */

    U32     fill_level;         /* bytes in use */

    U32     mode;               /* bit-flags: */
#define BM_ASF_MBOX_MODE_RD_ANYONE      (1<<0)  /* no auth req */
#define BM_ASF_MBOX_MODE_RD_OPERATOR    (1<<1)  /* auth required */
#define BM_ASF_MBOX_MODE_RD_ADMIN       (1<<2)  /* auth required */
#define BM_ASF_MBOX_MODE_WR_ANYONE      (1<<8)  /* no auth req */
#define BM_ASF_MBOX_MODE_WR_OPERATOR    (1<<9)  /* auth required */
#define BM_ASF_MBOX_MODE_WR_ADMIN       (1<<10) /* auth required */

    U32     owner;              /* IANA number, or 0 for nobody */
#define BM_ASF_MBOX_OWNER_ASF           4542    /* ASF IANA num */
#define BM_ASF_MBOX_OWNER_NOBODY        0
#define BM_ASF_MBOX_OWNER_BRCM          4413    /* Broadcom IANA enterprise num */

    /* remaining fields are "owner" defined: */
    U32     type;
#define BM_ASF_MBOX_TYPE_SMBIOS         0   /* ASF */

    U32     flags;
#define BM_ASF_MBOX_FLAGS_TAG_MASK      0x7     /* 8 possible tag methods (0-7) */
#define  BM_ASF_MBOX_FLAGS_TAG_USER       0        /*  tag is in user/application defined
format */
#define BM_ASF_MBOX_FLAGS_TAG_SEQNUM     1        /* tag is an incrementing sequence number
*/
#define BM_ASF_MBOX_FLAGS_TAG_CRC32     2       /* tag is calculated CRC-32 of contents */
#define BM_ASF_MBOX_FLAGS_TAG_TIME       3       /* tag is time_t date/time stamp of last
write */

    U32     tag;                /* time, digest, seqnum, etc. */

} BM_ASF_MBOX_STATUS;           /* 32 bytes */



/****************************************************************************/
/* Firmware information and definition                                      */
/*                                                                          */
/* ATTENTION:  Firmware is using Big-ENDIAN and Intel PC is using           */
/*            Little-ENDIAN.                                                */
/****************************************************************************/
#define BMAPI_NTOH32(p)             ( ( U32 )( (*((U8 *) (p)) << 24) | (*((U8 *) (p)+1) <<
16) | (*((U8 *) (p)+2) << 8) | *((U8 *) (p)+3) ) )
#define BMAPI_FW_MAGIC_VALUE            0x669955aa
#define BMAPI_FW_MEDIA_MAX_LOAD_STAGES  8

typedef struct _BM_FW_MEDIA_BOOTSTRAP_REGION
{
    // a pattern not likely to occur randomly
    U32     magic_value;

    // where to locate boot code (byte addr)
    U32     sram_start_addr;

    // boot code length (in words)
    U32     code_len;
    // location of code on media (media byte addr)
    U32     code_start_addr;

    // 32-bit CRC
    U32     cksum;

} BM_FW_MEDIA_BOOTSTRAP_REGION;


typedef enum {
    BMAPI_FW_CODE_IMAGE_TYPE_PXE,           /*  0 0x00 */
    BMAPI_FW_CODE_IMAGE_TYPE_ASF_INIT,      /*  1 0x01 */
    BMAPI_FW_CODE_IMAGE_TYPE_ASF_CPUA,      /*  2 0x02 */
    BMAPI_FW_CODE_IMAGE_TYPE_ASF_CPUB,      /*  3 0x03 */
    BMAPI_FW_CODE_IMAGE_TYPE_ASF_CFG,       /*  4 0x04 */
    BMAPI_FW_CODE_IMAGE_TYPE_ISCSI_CFG,     /*  5 0x05 */
    BMAPI_FW_CODE_IMAGE_TYPE_ISCSI_CFG_PRG, /*  6 0x06 */
    BMAPI_FW_CODE_IMAGE_TYPE_USER_BLOCK,    /*  7 0x07 */
    BMAPI_FW_CODE_IMAGE_TYPE_BRSF_BLOCK,    /*  8 0x08 */
    BMAPI_FW_CODE_IMAGE_TYPE_ISCSI_BOOT,    /*  9 0x09 */
    BMAPI_FW_CODE_IMAGE_TYPE_ASF_MBOX,      /* 10 0x0a */
    BMAPI_FW_CODE_IMAGE_TYPE_ISCSI_CFG_1,   /* 11 0x0b */
    BMAPI_FW_CODE_IMAGE_TYPE_APE_CFG,       /* 12 0x0c */
    BMAPI_FW_CODE_IMAGE_TYPE_APE_CODE,      /* 13 0x0d (APE firmware) */
    BMAPI_FW_CODE_IMAGE_TYPE_APE_UPDATE,    /* 14 0x0e (APE firmware update) */
    BMAPI_FW_CODE_IMAGE_TYPE_EXT_CFG,       /* 15 0x0f */
    BMAPI_FW_CODE_IMAGE_TYPE_EXT_DIR,       /* 16 0x10 */
    BMAPI_FW_CODE_IMAGE_TYPE_APE_DATA,        /* 17 0x11 (a.k.a. Opaque Management Data,
Offline Mailbox Data) */
    BMAPI_FW_CODE_IMAGE_TYPE_APE_WEB_DATA,  /* 18 0x12 (Web/HTML interface content) */
    BMAPI_FW_MAX_CODE_IMAGE_TYPE,
} BM_FW_CODE_IMAGE;


typedef struct _BM_FW_MEDIA_CODE_OFFSET_REGION
{
    // where to locate code region (byte addr)
    U32     sram_start_addr;

#define BMAPI_FW_CODE_LEN_RESERVED            0x00000000       /* all 8 bits are used for
types now */
#define BMAPI_FW_CODE_IMAGE_POS             24
#define BMAPI_FW_CODE_IMAGE_TYPE(x)          ((x >> BMAPI_FW_CODE_IMAGE_POS) & 0x00ff) /*
Use 8 bits -- supports up to 256 types */
#define BMAPI_FW_CODE_IMAGE_TYPE_MASK       0xff000000
#define BMAPI_FW_CODE_IMAGE_EXECUTE_A       0x00800000
#define BMAPI_FW_CODE_IMAGE_EXECUTE_B       0x00400000
#define BMAPI_FW_CODE_IMAGE_EXECUTABLE_A(x) (x & BMAPI_FW_CODE_IMAGE_EXECUTE_A)
#define BMAPI_FW_CODE_IMAGE_EXECUTABLE_B(x) (x & BMAPI_FW_CODE_IMAGE_EXECUTE_B)
#define BMAPI_FW_CODE_IMAGE_LENGTH(x)       (x & 0x3fffff)

    // code region length (in words)
    // The top 8 bits are used for TYPE field.
    // We use 22 bit for length so that we accumulate up to 16 Mbyte.
    U32     code_len;

    // location of code on media (media byte addr)
    U32     code_start_addr;

} BM_FW_MEDIA_CODE_OFFSET_REGION;


#define BMAPI_FW_FEATURE_CONFIG_PXE_SPEED_AUTO      0   // Auto
#define BMAPI_FW_FEATURE_CONFIG_PXE_SPEED_10_H      1   // 10Mbps Half Duplex
#define BMAPI_FW_FEATURE_CONFIG_PXE_SPEED_10_F      2   // 10Mbps Full Duplex
#define BMAPI_FW_FEATURE_CONFIG_PXE_SPEED_100_H     3   // 100Mbps Half Duplex
#define BMAPI_FW_FEATURE_CONFIG_PXE_SPEED_100_F     4   // 100Mbps Full Duplex
#define BMAPI_FW_FEATURE_CONFIG_PXE_SPEED_1000_H    5   // 1000Mbps Half Duplex
#define BMAPI_FW_FEATURE_CONFIG_PXE_SPEED_1000_F    6   // 1000Mbps Full Duplex

#define BMAPI_FW_FEATURE_CONFIG_BOOT_TYPE_PXE       0
#define BMAPI_FW_FEATURE_CONFIG_BOOT_TYPE_RPL       1
#define BMAPI_FW_FEATURE_CONFIG_BOOT_TYPE_BOOTP     2
#define BMAPI_FW_FEATURE_CONFIG_BOOT_TYPE_ISCSI     3

#define BMAPI_FW_FEATURE_CONFIG_BOOT_STRAP_AUTO     0
```

```
#define BMAPI_FW_FEATURE_CONFIG_BOOT_STRAP_BBS      1
#define BMAPI_FW_FEATURE_CONFIG_BOOT_STRAP_INT18    2
#define BMAPI_FW_FEATURE_CONFIG_BOOT_STRAP_INT19    3

#define BMAPI_FW_FEATURE_CONFIG_PXE_BAR_SIZE_64K    0
#define BMAPI_FW_FEATURE_CONFIG_PXE_BAR_SIZE_128K   1
#define BMAPI_FW_FEATURE_CONFIG_PXE_BAR_SIZE_256K   2
#define BMAPI_FW_FEATURE_CONFIG_PXE_BAR_SIZE_512K   3
#define BMAPI_FW_FEATURE_CONFIG_PXE_BAR_SIZE_1M     4
#define BMAPI_FW_FEATURE_CONFIG_PXE_BAR_SIZE_2M     5
#define BMAPI_FW_FEATURE_CONFIG_PXE_BAR_SIZE_4M     6
#define BMAPI_FW_FEATURE_CONFIG_PXE_BAR_SIZE_8M     7
#define BMAPI_FW_FEATURE_CONFIG_PXE_BAR_SIZE_16M    8

#define BMAPI_FW_WOL_MASK                   0x00000001
#define BMAPI_FW_PXE_MASK                   0x00000002
#define BMAPI_FW_MBA_MASK                   0x00000002
#define BMAPI_FW_PXE_SPEED_MASK             0x0000003c
#define BMAPI_FW_MBA_SPEED_MASK             0x0000003c
#define BMAPI_FW_MBA_TYPE_MASK              0x00300000
#define BMAPI_FW_MBA_PROT_PXE               0x00000000
#define BMAPI_FW_MBA_PROT_RPL               0x00100000
#define BMAPI_FW_MBA_PROT_BOOTP             0x00200000
#define BMAPI_FW_MBA_PROT_ISCSI             0x00300000
#define BMAPI_FW_MBA_TYPE_SHIFT_POS         20
#define BMAPI_FW_FORCE_PCI_MODE_MASK        0x00000040
#define BMAPI_FW_ASF_MASK                   0x00000080
#define BMAPI_FW_PXE_BAR_SIZE               0x00000f00
#define BMAPI_FW_EEPROM_WP_MASK             BIT_22
#define BMAPI_FW_EEPROM_WP_SHIFT_POS        22
#define BMAPI_FW_WOL_SPEED_LOMIT10          BIT_25
#define BMAPI_FW_LINK_IDLE_MODE_ENABLE      BIT_26
#define BMAPI_FW_CABLE_SENSE_ENABLE         BIT_28
#define BMAPI_LINK_AWARE_MODE_ENABLE        BIT_30
#define BMAPI_LINK_SPEED_POWER_MODE_ENABLE  BIT_31

#define BMAPI_FW_PXE_BOOT_TYPE_MASK         0x00300000
#define BMAPI_FW_BOOT_TYPE_PXE_SHIFT_POS    20
#define  BMAPI_FW_PXE_BOOT_TYPE_PXE              (BMAPI_FW_FEATURE_CONFIG_BOOT_TYPE_PXE  <<
BMAPI_FW_BOOT_TYPE_PXE_SHIFT_POS)
#define  BMAPI_FW_PXE_BOOT_TYPE_RPL              (BMAPI_FW_FEATURE_CONFIG_BOOT_TYPE_RPL  <<
BMAPI_FW_BOOT_TYPE_PXE_SHIFT_POS)
#define  BMAPI_FW_PXE_BOOT_TYPE_BOOTP          (BMAPI_FW_FEATURE_CONFIG_BOOT_TYPE_BOOTP <<
BMAPI_FW_BOOT_TYPE_PXE_SHIFT_POS)

#define BMAPI_FW_PXE_BAR_SIZE_SHIFT_POS     8

#define BMAPI_VAUX_CUTOFF_DELAY_CHOICE(x)   (((x)>>23) & 0x3)

#define BMAPI_DRIVER_WOL_DISABLE            0
#define BMAPI_DRIVER_WOL_MAGIC_ENABLE       1
#define BMAPI_DRIVER_WOL_INT_PKT_ENABLE     2
#define BMAPI_DRIVER_WOL_BOTH_ENABLE        3



typedef union _BM_FW_FEATURE_CONFIG
{
    struct
    {
#if defined(BIG_ENDIAN) && !defined(BCM_LITTLE_ENDIAN_HOST)
        U32 link_speed_power    :1;     /* link speed power mode for Taishan */
        U32 link_aware          :1;     /* link aware mode for Taishan */
        U32 mba_vlan_enable     :1;     /* bit 29 - Enable VLAN in Multiple Boot Agent */
        U32 cable_sense         :1;
       U32  reserved             :1;      /*  bit  27       */   /*driver_wol_enable  is  no
longer used by driver and redefined */
       U32  link_idle            :1;      /*  bit  26       */   /*driver_wol_enable  is  no
longer used by driver and redefined */
        U32 Wol_Limit_10        :1;
        U32 vaux_cutoff_delay   :2;
        U32 lom_design          :1;
        U32 pxe_boot_protocol   :2;
        U32 pxe_timeout_msg     :4;
        U32 pxe_bootstrap_type  :2;     /* This field should be used by */
                                        /* boot code. This is only */
                                        /* referenced by MBA. */
        U32 hot_key_option       :1;      /* bit 13 - configure hot-key option: Ctrl-S or
Ctrl-B  */
        U32  disable_setup_msg     :1;        /*  bit  12  -  Disable  setup  prompt  for  MBA
configuration */
        U32 pxe_bar_size        :4;
        U32 asf_enable          :1;
        U32 force_pci           :1;
        U32 pxe_speed           :4;
        U32 pxe_enable          :1;
        U32 wol_enable          :1;
#else
        U32 wol_enable          :1;     /* bit 0      */
        U32 pxe_enable          :1;     /* bit 1      */
        U32 pxe_speed           :4;     /* bit 2:5    */
        U32 force_pci           :1;     /* bit 6      */
        U32 asf_enable          :1;     /* bit 7      */
        U32 pxe_bar_size        :4;     /* bit 8:11   */
        U32  disable_setup_msg     :1;        /*  bit  12  -  Disable  setup  prompt  for  MBA
configuration */
        U32 hot_key_option       :1;      /* bit 13 - configure hot-key option: Ctrl-S or
Ctrl-B  */
        U32 pxe_bootstrap_type  :2;     /* bit 14:15  */
        U32 pxe_timeout_msg     :4;     /* bit 16:19  */
        U32 pxe_boot_protocol   :2;     /* bit 20:21  */ /* This field */
                                        /* should be used by boot code. */
                                        /* This is only referenced by MBA. */
        U32 lom_design          :1;     /* bit 22     */
        U32 vaux_cutoff_delay   :2;     /* bit 23:24  */
        U32 Wol_Limit_10        :1;     /* bit 25     */
        U32  link_idle             :1;      /*  bit  26       */   /*driver_wol_enable  is  no
longer used by driver and redefined */
        U32  reserved              :1;      /*  bit  27       */   /*driver_wol_enable  is  no
longer used by driver and redefined */
        U32 cable_sense         :1;     /* bit 28     */
        U32 mba_vlan_enable     :1;     /* bit 29 - Enable VLAN in Multiple Boot Agent */
        U32 link_aware          :1;     /* for Taishan */
        U32 link_speed_power    :1;     /* link speed power mode for Taishan */
#endif
    } bit;

    U32 word;
} BM_FW_FEATURE_CONFIG;



#define BMAPI_FW_NIC_HW_CONFIG_PHY_TYPE_UNKNOWN             0
#define BMAPI_FW_NIC_HW_CONFIG_PHY_TYPE_COPPER              1
#define BMAPI_FW_NIC_HW_CONFIG_PHY_TYPE_FIBER               2

/* the defines for mode setting in menu */
#define BMAPI_FW_NIC_HW_CONFIG_LED_MAC_MODE                 0
#define BMAPI_FW_NIC_HW_CONFIG_LED_MODE_PHY_MODE_1          1
#define BMAPI_FW_NIC_HW_CONFIG_LED_MODE_PHY_MODE_2          2
#define BMAPI_FW_NIC_HW_CONFIG_LED_MODE_SHARED_TRAFFIC      3
#define BMAPI_FW_NIC_HW_CONFIG_LED_MODE_SHASTA_MAC_MODE     4
#define BMAPI_FW_NIC_HW_CONFIG_LED_MODE_WIRELESS_COMBO_MODE 5

#define BMAPI_FW_NIC_HW_CONFIG_LED_MODE_UNKNOWN             0
#define BMAPI_FW_NIC_HW_CONFIG_LED_MODE_TRIPLE_LINK         1
#define BMAPI_FW_NIC_HW_CONFIG_LED_MODE_LINK_SPEED          2

#define BMAPI_FW_NIC_HW_CONFIG_VOLTAGE_1_3_V        0x0
#define BMAPI_FW_NIC_HW_CONFIG_VOLTAGE_1_8_V        0x1
#define BMAPI_FW_NIC_HW_CONFIG_VOLTAGE_MASK         0x3
#define BMAPI_FW_NIC_HW_CONFIG_VOLTAGE_PARM(x)      (x & NIC_HW_CONFIG_VOLTAGE_MASK)

/* Both channel A and B are used */
#define BMAPI_FW_NIC_HW_CONFIG_DUAL_MAC_NORMAL      0x0
/* Only Channel B is used */
#define BMAPI_FW_NIC_HW_CONFIG_DUAL_MAC_CHAN_B      0x1
/* Only Channel A is used */
#define BMAPI_FW_NIC_HW_CONFIG_DUAL_MAC_CHAN_A      0x2
/* Use XBAR. single config. space for both channels. */
#define BMAPI_FW_NIC_HW_CONFIG_DUAL_MAC_XBAR        0x3
```

```
#define BMAPI_FW_NIC_HW_CONFIG_DAUL_MAC_MASK        0xc00
#define BMAPI_FW_NIC_DUAL_MAC_SHIFT_POS             10
#define BMAPI_FW_NIC_HW_CONFIG_DUAL_MAC_PARM(x)      ((x & NIC_HW_CONFIG_DAUL_MAC_MASK) >>
NIC_DUAL_MAC_SHIFT_POS)

#define BMAPI_FW_SHASTA_EXT_LED_LEGACY_MODE                     0
#define BMAPI_FW_SHASTA_EXT_LED_SHARED_TRAFFIC_LINK_LED_MODE    1
#define BMAPI_FW_SHASTA_EXT_LED_SHASTA_MAC_MODE                 2
#define BMAPI_FW_SHASTA_EXT_LED_WIRELESS_COMBO_MODE             3

#define BMAPI_FW_HW_CFG_VOLTAGE_SOURCE_MASK         (BIT_0 | BIT_1)
#define BMAPI_FW_HW_CFG_PHY_LED_MODE_MASK           (BIT_2 | BIT_3)
#define BMAPI_FW_HW_CFG_PHY_TYPE_MASK               (BIT_4 | BIT_5)
#define BMAPI_FW_HW_CFG_FORCE_MAX_PCI_RETRY         BIT_6
#define BMAPI_FW_HW_CFG_MAX_PCI_RETRY_MASK          (BIT_7 | BIT_8 | BIT_9)
#define BMAPI_FW_HW_CFG_DUAL_MAC_MODE_MASK          (BIT_10 | BIT_11)
#define BMAPI_FW_HW_CFG_REVERSE_WAY                 BIT_12
#define BMAPI_FW_HW_CFG_MINI_PCI                    BIT_13
#define BMAPI_FW_HW_CFG_AUTO_POWERDOWN              BIT_14
#define BMAPI_FW_HW_CFG_EXT_LED_MODE_MASK           (BIT_15 | BIT_16)
#define BMAPI_FW_HW_CFG_CAPACITIVE_COUPLING         BIT_17
#define BMAPI_FW_HW_CFG_TXSERDES_OVERRIDE           BIT_18
#define BMAPI_FW_HW_CFG_L0s_PERFORMANCE_FIX_EN      BIT_19

#define BMAPI_FW_CAPACITIVE_COUPLING                BMAPI_FW_HW_CFG_CAPACITIVE_COUPLING



typedef union _BM_FW_NIC_HW_CONFIG
{
    struct
    {
#if defined(BIG_ENDIAN) && !defined(BCM_LITTLE_ENDIAN_HOST)
        U32  reserved                       :13;      /*  bit  19         */  /*  used  to  be
ASPM_L1,ASPM_L0 and clkreq at bit 21, 20 and 19 */
        U32 txSerdesOverride        :1;     /* bit 18      */
        U32 capacitive_coupling     :1;     /* bit 17      */
        U32 shasta_ext_led_mode     :2;     /* bit 15,16   */
        U32 enable_auto_powerdown   :1;     /* bit 14      */
        U32 mini_pci                :1;     /* bit 13      */
        U32 reverse_nway            :1;     /* bit 12      */
        U32 dual_mac_mode           :2;     /* bit 10:11   */
        U32 max_pci_retry           :3;     /* bit 7:9     */
        U32 forced_max_pci_retry    :1;     /* bit 6       */
        U32 phy_type                :2;     /* bit 4:5     */
        U32 phy_led_mode            :2;     /* bit 2:3     */
        U32 voltage_source          :2;     /* bit 0:1     */
#else
        U32 voltage_source          :2;     /* bit 0:1     */
        U32 phy_led_mode            :2;     /* bit 2:3     */
        U32 phy_type                :2;     /* bit 4:5     */
        U32 forced_max_pci_retry    :1;     /* bit 6       */
        U32 max_pci_retry           :3;     /* bit 7:9     */
        U32 dual_mac_mode           :2;     /* bit 10:11   */
        U32 reverse_nway            :1;     /* bit 12      */
        U32 mini_pci                :1;     /* bit 13      */
        U32 enable_auto_powerdown   :1;     /* bit 14      */
        U32 shasta_ext_led_mode     :2;     /* bit 15,16   */
        U32 capacitive_coupling     :1;     /* bit 17      */
        U32 txSerdesOverride        :1;     /* bit 18      */
        U32  reserved                       :13;      /*  bit  19         */  /*  used  to  be
ASPM_L1,ASPM_L0 and clkreq at bit 21, 20 and 19 */
#endif
    } bit;

    U32     word;
} BM_FW_NIC_HW_CONFIG;



typedef union _BM_NIC_SHARED_CONFIG
{
    struct
    {
#if defined(BIG_ENDIAN) && !defined(BCM_LITTLE_ENDIAN_HOST)
        U32 reserved: 15;           /* bit 17:31 */
        U32 gpio2_output_value:1;   /* bit 16 */
        U32 reserved_15:1;          /* bit 15 -- reserved */
        U32 gpio0_output_value:1;   /* bit 14 */
        U32 gpio2_output_enable:1;  /* bit 13 */
        U32 reserved_12:1;          /* bit 12 -- reserved */
        U32 gpio0_output_enable:1;  /* bit 11 */
        U32 reserved_7_10:4;        /* bit 7:10 -- reserved */
        U32 HotPlugPwrBdgtCnt:3;    /* bit 4:6 */
        U32 DisablePowerSaving:1;   /* bit 3 */
        U32 FiberWoLCapable:1;      /* bit 2 */
        U32 L1ASPM_Debounce_En:1;    /* bit 1 it was BothPort100MbpsCapable, now become
L1ASPM_Debounce_En*/
        U32 portSwap:1;             /* bit 0 */
#else
        U32 portSwap:1;             /* bit 0 */
        U32 L1ASPM_Debounce_En:1;    /* bit 1 it was BothPort100MbpsCapable, now become
L1ASPM_Debounce_En*/
        U32 FiberWoLCapable:1;      /* bit 2 */
        U32 DisablePowerSaving:1;   /* bit 3 */
        U32 HotPlugPwrBdgtCnt:3;    /* bit 4:6 */
        U32 reserved_7_10:4;        /* bit 7:10 -- reserved */
        U32 gpio0_output_enable:1;  /* bit 11 */
        U32 reserved_12:1;          /* bit 12 -- reserved */
        U32 gpio2_output_enable:1;  /* bit 13 */
        U32 gpio0_output_value:1;   /* bit 14 */
        U32 reserved_15:1;          /* bit 15 -- reserved */
        U32 gpio2_output_value:1;   /* bit 16 */
        U32 reserved: 15;           /* bit 17:31 */
#endif
    } bit;

    U32 word;
} BM_NIC_SHARED_CONFIG;

#define BMAPI_FW_SH_CFG_HOTPLUG_PWRBDGT_CNT_MASK    0x00000070
#define BMAPI_FW_SH_CFG_DIS_PW_SAVING               BIT_3
#define BMAPI_FW_SH_CFG_GPIO_MASK                   0x16800



typedef struct _BM_FW_MEDIA_MANUFACT_REGION
{
    // Version format of this data structure
    U8                      manuf_format_rev;

    // Checksum of directory section. When this byte
    // is zero, the checksum is not calculated.  When
    // this byte is non-zero, adding all bytes from 0x14-0x73
    // and this byte should be zero.
    U8                      dir_cksum;

    // length of this structure in bytes
    U16                     length;

    // formerly, it was PHY ID of physical device, no longer in use since 5/1/03
    U32                     reserved2;

    // MAC address
    U8                      mac_address[8];

    // part number. Printable string ending in '\0'.
    U8                      part_number[16];

    // part revision. Two printable ascii char.
    U8                      part_revision[2];

    // single byte acending rev number
    // Lower byte is major version number. (Little-Endian)
    // Higher byte is minor version number. (Little-Endian)
    U16                     bootcode_fw_revision;

    // ascii manufact date, wwyy, \0 by next field
    U8                      manuf_data[4];

    // VLAN ID used in Multiple Boot Agent - Port 1
    U16                     mba_vlan_id;
    // VLAN ID used in Multiple Boot Agent - Port 2
    U16                     mba_vlan_id_b;

    // pci device id
    U16                     pci_dev_id;

    // pci vendor id
    U16                     pci_vend_id;

    // alternate pci subsystem id
    U16                     pci_sub_id;

    // alternate pci subsystem vendor id
    U16                     pci_sub_vend_id;

    // cpu clock speed in Mhz rounded
    U16                     cpu_clk;

    // Port 1 smbus address
    U8                      smbus_addr1;

    // Port 0 smbus address
    U8                      smbus_addr0;

    // Backup permanent MAC address of port 1
    U8                      backup_mac_address[8];

    // Backup permanent MAC address of port 2
    U8                      backup_mac_addressb[8];

    U8                      power_dissipated[4];
    U8                      power_consumed[4];

    BM_FW_FEATURE_CONFIG    feature_config;

    BM_FW_NIC_HW_CONFIG     hw_config;

    // Second MAC address for second channel in BCM5704
    U8                      mac_addressb[8];

    // second MAC feature_config
    BM_FW_FEATURE_CONFIG    feature_configb;

    // second MAC hw_config
    BM_FW_NIC_HW_CONFIG     hw_configb;

    BM_NIC_SHARED_CONFIG    shared_config;

    U32                     Power_Budget0;
    U32                     Power_Budget1;

    U32                     serworks_use;

    /* Primary MAC Serdes 0:15 override value */
    U16                     mac0_txSerdes_value;

    /* Secondary MAC Serdes 0:15 override value */
    U16                     mac1_txSerdes_value;

    /* size in k (2^10). value 0 = unknown */
    U16                     tpm_nvram_size;

    /* size in k (2^10). value 0 = unknown */
    U16                     mac_nvram_size;

    U32                     Power_Budget2;
    U32                     Power_Budget3;

    // 32-bit CRC
    U32                     cksum;

} BM_FW_MEDIA_MANUFACT_REGION;


#define BMAPI_FW_MAX_VPD_R_LENGTH           128
```

```
#define BMAPI_FW_MAX_VPD_W_LENGTH           128

typedef struct _BM_FW_MEDIA_VPD_READ
{
    U8  data[BMAPI_FW_MAX_VPD_R_LENGTH];
} BM_FW_MEDIA_VPD_READ;


typedef struct _BM_FW_MEDIA_VPD_WRITE
{
    U8  data[BMAPI_FW_MAX_VPD_W_LENGTH];
} BM_FW_MEDIA_VPD_WRITE;


typedef struct _BM_FW_MEDIA_VPD
{
    BM_FW_MEDIA_VPD_READ    vpd_r;
    BM_FW_MEDIA_VPD_WRITE   vpd_w;
} BM_FW_MEDIA_VPD;



typedef struct _BM_FW_EEPROM_INFO
{
    BM_FW_MEDIA_BOOTSTRAP_REGION    bootstrap;
    BM_FW_MEDIA_CODE_OFFSET_REGION  code_offsets[BMAPI_FW_MEDIA_MAX_LOAD_STAGES];
    BM_FW_MEDIA_MANUFACT_REGION     manufact;
    BM_FW_MEDIA_VPD                 vpd;

} BM_FW_EEPROM_INFO;


#define BMAPI_FW_INFO_VER   7

typedef struct _BM_FW_INFO
{
    //  Version is defined as BMAPI_FW_INFO_VER.
    //  'version' is required upon input
    U32                 version;

    // Not valid if the NIC is set for selfboot.
    // Use 'is_selfboot' to find out the NIC selfboot or not.
    BM_FW_EEPROM_INFO   eeprom_info;

    // Boot code version description string.
    U8                  firmware_ver_desc[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // PXE version description string.
    // Not valid if the NIC is set for selfboot.
    // Use 'is_selfboot' to find out the NIC selfboot or not.
    U8                  pxe_ver_desc[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // Boolean flag to indicate whether the NIC is SW selfboot or not.
    U32                 is_selfboot;

    // '0' for SW selfboot format 0, '1' for SW selfboot format 1, etc.
    // Not valid if the NIC is NOT selfboot.
    U32                 selfboot_format;

    // iSCSI versions.
    U8                  iscsi[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  iscsi_cprg[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // Boolean flag to indicate whether the NIC is HW selfboot or not.
    U32                 is_hw_selfboot;

    // Management FW version. Can be ASF, IPMI, UMP or DASH.
    // Use 'flags' to determine which one it is.
    U8                  mgmt_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

#define BM_MGMT_ASF             0x00000001  // ASF Management Firmware
#define BM_MGMT_ASF_2_0             0x00000002  // ASF Management Firmware with ASF 2.0
support
#define BM_MGMT_IPMI            0x00000004  // IPMI Management Firmware
#define BM_MGMT_UMP             0x00000008  // UMP Management Firmware
#define BM_MGMT_DASH            0x00000010  // DASH Management Firmware
```

```
    U32                 flags;

    // Boolean flag to indicate whether the NIC is OTP selfboot or not.
    U32                 is_otp_selfboot;

} BM_FW_INFO;


#define BMAPI_FW_INFO_5706_VER      5

typedef struct _BM_FW_INFO_5706
{
    //  Version is defined as BMAPI_FW_INFO_5706_VER.
    //  'version' is required upon input
    U32                 version;

    // Boot code version string.
    U8                  bc1_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  bc2_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // MBA/PXE version string.
    U8                  pxe_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // L2RXP version string.
    U8                  l2rxp_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // UMP version string.
    U8                  ump_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // IPMI versions.
    U8                  ipmi_cfg[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  ipmi_init[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  ipmi_serv[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // iSCSI versions.
    U8                  iscsi[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  iscsi_cprg[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // VPD data
    BM_FW_MEDIA_VPD     vpd;

    // iSCSI cfg versions.
    U8                  iscsib_cfg[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  iscsib_cfg2[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // NCSI versions.
    U8                  ncsi_common[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  ncsi_lib5706[BMAPI_FW_MAX_DESCRIPTION_LEN];

} BM_FW_INFO_5706;



#define BMAPI_FW_INFO_57710_VER      4

typedef struct _BM_FW_INFO_57710
{
    //  Version is defined as BMAPI_FW_INFO_57710_VER.
    //  'version' is required upon input
    U32                 version;

    // Boot code version string.
    U8                  bc1_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  bc2_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // MBA/PXE version string.
    U8                  pxe_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // UMP version string.
    U8                  ump_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // IPMI versions.
    U8                  ipmi_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // iSCSI versions.
    U8                  iscsib[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  iscsi_cprg[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  iscsib_cfg[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  iscsib_cfg2[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // NCSI versions. This is the lib version for 57710 (as ncsi_lib5706
    // for 5706).
    U8                  ncsi_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // TSTORM versions.
    U8                  l2t_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // CSTORM versions.
    U8                  l2c_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // XSTORM versions.
    U8                  l2x_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // USTORM versions.
    U8                  l2u_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // VPD data
    BM_FW_MEDIA_VPD     vpd;

    // External PHY version.
    U8                  ext_phy_fw_ver[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // NCSI versions.
    U8                  ncsi_common[BMAPI_FW_MAX_DESCRIPTION_LEN];

    // FCoE versions.
    U8                  fcoe_boot[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  fcoe_cprg[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  fcoe_cfg[BMAPI_FW_MAX_DESCRIPTION_LEN];
    U8                  fcoe_cfg2[BMAPI_FW_MAX_DESCRIPTION_LEN];

} BM_FW_INFO_57710;


/*****************************************************************************
 *
 *                              ATTENTION
 *
 * Multi-byte fields in BMAPI_LICENSE_INFO such as 'capability',
 * 'max_toe_conn', 'sn', etc. are in BIG_ENDIAN order. For LITTLE-ENDIAN
 * macine such as 'x86' processor, the bytes for the multi-byte data fields
 * MUST be swapped.
 *****************************************************************************/
// 'key_index' for BmapiGetLicenseKey() and BmapiSetLicenseKey().
#define BMAPI_LICENSE_KEY_IDX_ACTIVE             0   // The key that is currently used by
firmware
#define BMAPI_LICENSE_KEY_IDX_MANUFAC           1
#define BMAPI_LICENSE_KEY_IDX_UPGRADE           2

typedef struct
{
#define BMAPI_LICENSE_KEY_TYPE_ENUM_BCM5706     0x0
#define BMAPI_LICENSE_KEY_TYPE_ENUM_BCM57710    0x2
      U8      key_type;

#define BMAPI_LICENSE_INFO_VER                  1
      U8      version;

      U8      dword_length;     /* Not including the digest */

#define BMAPI_LICENSE_OEM_ID_BRCM               0
      U8      oem_id;

#define BMAPI_LICENSE_CAP_USER_RDMA             0x0002  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_TOE                   0x0004  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_ISCSI_INIT            0x0008  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_ISCSI_TRGT            0x0010  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_ISER_INIT             0x0020  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_ISER_TRGT             0x0040  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_ISCSI_BOOT            0x0080  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_ISCSI_FULL_ACCL       0x0100  /* BCM5706 only */
```

```
#define BMAPI_LICENSE_CAP_ISCSI_HDR_DGST        0x0200  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_ISCSI_BODY_DGST       0x0400  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_SERDES_2_5G           0x0800  /* BCM5706 only */
#define BMAPI_LICENSE_CAP_SPEED_12G             0x0800  /* BCM57710 Only */
#define BMAPI_LICENSE_CAP_SPEED_12_5G           0x1000  /* BCM57710 Only */
#define BMAPI_LICENSE_CAP_SPEED_13G             0x2000  /* BCM57710 Only */
#define BMAPI_LICENSE_CAP_SPEED_15G             0x4000  /* BCM57710 Only */
#define BMAPI_LICENSE_CAP_SPEED_16G             0x8000  /* BCM57710 Only */
      U16     capability;

#define BMAPI_LICENSE_CONN_UNLIMITED            0xffff
      U16     max_toe_conn;

      U16     reserved;
      U16     max_um_rdma_conn;
      U16     max_iscsi_init_conn;
      U16     max_iscsi_trgt_conn;
      U16     max_iser_init_conn;
      U16     max_iser_trgt_conn;

      U32     reserved_a[3];
      U32     sn;
      U16     reserved_b;

#define BMAPI_LICENSE_EXP_NEVER                 0xffff
      U16     expiration;

} BMAPI_LICENSE_INFO;



#define BMAPI_RES_CFG_VER         5

typedef struct
{
    // Version is defined as BMAPI_RES_CFG_VER.
    // 'version' is required upon input
    // All application MUST use verison 2 or later structure.
    U32 version;

#define BMAPI_RES_RES_CFG_VALID                 0x01        // for BMAPI_BRCM5706 only
#define BMAPI_RES_CFG_L2                        0x04
#define BMAPI_RES_CFG_ISCSI                     0x08
#define BMAPI_RES_CFG_RDMA                      0x10
#define BMAPI_RES_CFG_FCFS_DISABLED             0x80000000L // for BMAPI_BRCM5706 only
    // The 'cfg' is to dictate which child nodes driver is configured to
    // (when getting) or will (when setting) enumerate.
    U32 cfg;

    // When IPV6 is set, the # of connections meant for IPV6.
    // If it is clear, the number meant for IPV4. This is for
    // 'BMAPI_BRCM5706' only.
#define BMAPI_RES_RES_CFG_TOE_IPV6              (0x1 << 0)  // for BMAPI_BRCM5706 only
#define BMAPI_RES_RES_CFG_ISCSI_IPV6            (0x1 << 1)  // for BMAPI_BRCM5706 only
    U32 flags;

    U32 reserved1;
    U32 reserved2;

    // For BMAPI_BRCM5706 and BMAPI_BRCM57710
    U32 toe_licensed_con;       // number of licensed TOE connections

    // For BMAPI_BRCM5706
    U32 toe_reserved_con;       // number of reserved TOE connections

    U32 reserved3;
    U32 reserved4;

    // For BMAPI_BRCM5706 and BMAPI_BRCM57710
    U32 rdma_licensed_con;      // number of licensed RDMA connections

    // For BMAPI_BRCM5706
    U32 rdma_reserved_con;      // number of reserved RDMA connections

    U32 reserved5;
    U32 reserved6;
    // For BMAPI_BRCM5706 and BMAPI_BRCM57710
    U32 iscsi_licensed_con;     // number of licensed iSCSI connections

    // For BMAPI_BRCM5706
    U32 iscsi_reserved_con;     // number of reserved iSCSI connections

    U32 reserved7;
    U32 reserved8;

    // For BMAPI_BRCM5706 and BMAPI_BRCM57710
    U32 iser_licensed_con;      // number of licensed iSER connections

    // For BMAPI_BRCM5706
    U32 iser_reserved_con;      // number of reserved iSER connections

    // For BMAPI_BRCM5706 and BMAPI_BRCM57710
    U32 iscsi_pend_task;        // iSCSI pending tasks per connection
                                // BMAPI_BRCM5706: range from 32 to 2048
                                // BMAPI_BRCM57710: range from 64 to 2048
                                // It should be power of 2.

    //////////////////////////
    // For BMAPI_BRCM57710 ONLY
    U32 ring_size;
    U32 max_toe_cons;
    U32 max_rdma_cons;          // Not supported currently
    U32 max_iscsi_cons;

    U32 toe_cons_limit;

    U32 toe_rss_possible;
    //////////////////////////

} BM_RES_CFG;



/****************************************************************************/
// Definition for BMAPI_CAB_DIAG
#define BMAPI_NUMCHAN                           4

#define BMAPI_CAB_DIAG_VER      1

typedef struct
{
    //  Version is defined as BMAPI_CABLE_DIAG_VER.
    //  'version' is required upon input
    U32 version;

    // 'status' definitions
#define BMAPI_CABDIAG_STATUS_NO_FAULT             0x0000  // Good cable/PCB signal paths,
but no Gigabit link, vlaid length returned
#define BMAPI_CABDIAG_STATUS_PIN_SHORT_OR_XT    0x0001  // Pin-short or cross-talk along 2
or more cable/PCB signal paths, vlaid length returned
#define BMAPI_CABDIAG_STATUS_OPEN              0x0002  // One or both pins are open for a
twisted pair, vlaid length returned
#define BMAPI_CABDIAG_STATUS_SHORT              0x0004  // Two pins from the same twisted
pair are shorted together, vlaid length returned
#define BMAPI_CABDIAG_STATUS_FORCED             0x0008  // Persistent noise present (most
likely caused by Forced 10/100), distanceCm is NOT valid
#define BMAPI_CABDIAG_STATUS_GLINK             0x0010  // Gigabit link is up and running,
vlaid length returned
#define BMAPI_CABDIAG_STATUS_UNKNOWN             -99      // unknow result, distanceCm is
NOT valid
    // 'status' determine the result condition of the able and whether
    // 'distanceCm' is valid or not.
    int status[BMAPI_NUMCHAN];

    // length of cable (in centimeters)
    U32 distanceCm[BMAPI_NUMCHAN];

} BMAPI_CAB_DIAG;

/****************************************************************************/
```

```
#define BMAPI_OFLD_STACK_INFO_NDIS  0
#define BMAPI_OFLD_STACK_INFO_ISCSI 2
#define BMAPI_OFLD_STACK_INFO_WSD   3

#define BMAPI_OFLD_STACK_INFO_VER   1

typedef struct
{
    //  Version is defined as BMAPI_OFLD_STACK_INFO_VER.
    //  'version' is required upon input
    U32 version;

    // the numbers of connections offloaded and connections being offloaded
    // but not completed yet
    U32 tcp_list_cnt;

    U32 reserved[18];

} BMAPI_OFLD_STACK_INFO;



#define BMAPI_ISCSI_CONFIG_VER      3

typedef struct
{
    //  Version is defined as BMAPI_ISCSI_CONFIG_VER.
    //  'version' is required on input
    U32 version;

    // 1 to enable DHCP for IPv4, 0 for static IP settings.
    // When DHCP is in use, 'LocalIpv4', 'SubnetMaskIpv4' and
    // 'DefaultGatewayIpv4' will be ignored.
    U32 EnableDHCPIpv4;

    // IPv4 configuration
    U32 LocalIpv4;
    U32 SubnetMaskIpv4;
    U32 DefaultGatewayIpv4;

    U32 VlanID;

    // Process Router Advertisements (BOOLEAN)
    U32 bProcRouteAd;

    // DHCPv6 (BOOLEAN)
    U32 bEnableDHCPIpv6;

    U32 mtu;

} BMAPI_ISCSI_CONFIG;



typedef struct
{
    U8  ipv6[16];
    U32 prefixlen;
} BM_ISCIS_IPV6_ADDR;



typedef struct
{
    // handle to an iSCSI node.
    // Must be privided on input.
    U32 handle;

    U32 uNumOfIPV4;
    U32 uNumOfIPV6;

    // Return code for this structure.
    // Applications must check BmapiGetISCSIRuntimeIPCount() to be BMAPI_OK
    // before using 'return_code' from each individual structure.
    U32 return_code;

} BM_ISCSI_IP_COUNT;



#define BM_ISCSI_IPV4_RT_VER        1

typedef struct
{
    //  Version is defined as BM_ISCSI_IPV4_RT_VER.
    //  'version' is required upon input
    U32 version;

    // handle to an iSCSI node.
    U32 handle;

    // 'flags' should set to '0' on input.
#define  BMAPI_ISCSI_IPV4_ADDR_INIT         0x0001               //  IPv4  address  has  been
initialized
#define BMAPI_ISCSI_IPV4_ADDR_VALID     0x0002          // IPv4 address is valid
#define BMAPI_ISCSI_IPV4_HBA_BOOT        0x0004           // HBA boot through 'this' IPV4
address
    U32 flags;

    // This conains the binary IPv4 ip address
    U32 IpV4Address;

    // TRUE if the adapter should use DHCP to discovery its IP address
    // information.
    U32 EnableDHCP;

    // Static Default Gateway IP address
    U32 DefaultGateway;

    // Static Subnet Mask
    U32 SubnetMask;

    // Return code for this structure.
    // Applications must check BmapiRetrieveMultiLinkStatus() to be BMAPI_OK
    // before using 'return_code' from each individual structure.
    U32 return_code;

} BM_ISCSI_IPV4_RT;



#define BM_ISCSI_IPV6_RT_VER        1

typedef struct
{
    //  Version is defined as BM_ISCSI_IPV6_RT_VER.
    //  'version' is required upon input
    U32 version;

    // handle to an iSCSI node.
    U32 handle;

    // TRUE if the adapter should use an autogenerated and non routable
    // (link local) address as its IP address.
    U32 UseLinkLocalAddress;

    // 'flags' should set to '0' on input.
#define BMAPI_ISCSI_IPV6_ADDR_INIT      0x0001          // IPv6 struct init happened
#define BMAPI_ISCSI_IPV6_ADDR_VALID     0x0002          // IPv6 address is valid
#define BMAPI_ISCSI_IPV6_GTW_VALID      0x0004          // IPv6 gateway is valid
#define BMAPI_ISCSI_IPV6_MASK_VALID     0x0008          // IPv6 subnet mask is valid
#define BMAPI_ISCSI_IPV6_HBA_BOOT        0x0010           // HBA boot through 'this' IPV6
address
    U32 flags;

    // TRUE if the adapter should use DHCP to discovery its IP address
    // information.
    U32 EnableDHCP;

    // IPv6 address of the adapter
    U8  IpV6Address[16];

    // IPV6 flow information
    U32 FlowInfo;

    // IPV6 scope id
    U32 ScopeId;

    // Static Default Gateway IP address
    U8  DefaultGateway[16];

    // Static Subnet Mask
    U8  SubnetMask[16];

    // Return code for this structure.
    // Applications must check BmapiRetrieveMultiLinkStatus() to be BMAPI_OK
    // before using 'return_code' from each individual structure.
    U32 return_code;

} BM_ISCSI_IPV6_RT;



#define BM_ISCSI_STATS_VER        1

typedef struct
{
    //  Version is defined as BM_ISCSI_STATS_VER.
    //  'version' is required upon input
    U32 version;

    // handle to an iSCSI node.
    U32 handle;

    // Unique Adapter Id
    U64 UniqueAdapterId;

    // Number of sessions
    U32 SessionCount;

    ////////////////////////////////////
    //////Initiator Login Statistics////
    ////////////////////////////////////
    // Count of Login Accept Responses
    U32 LoginAcceptRsps;

    // Count of Login other failed Responses
    U32 LoginOtherFailRsps;

    // Count of Login Redirect Responses
    U32 LoginRedirectRsps;

    // Count of Login Authentication Failed Responses
    U32 LoginAuthFailRsps;

    // Count of the number of times a login is aborted
    // due to a target authentication failure
    U32 LoginAuthenticateFails;

    // Count of the number of times login failed due
    // to negotiation failure with target
    U32 LoginNegotiateFails;

    // Count of Logout command PDU with reason code 0
    U32 LogoutNormals;

    // Count of Logout command PDUs with status code other than 0
    U32 LogoutOtherCodes;

    // The object counts the number of times a login
    // attempt from this local initiator has failed
    U32 LoginFailures;

    /////////////////////////////////////////
    //////Initiator Instance Statistics//////
    /////////////////////////////////////////
    // Count of Session digest errors
    U32 SessionDigestErrorCount;

    // Count of Session connection timeout error
    U32 SessionConnectionTimeoutErrorCount;

    // Count of Session format error
    U32 SessionFormatErrorCount;

    // Number of Sessions failed belonging to this instance
    U32 SessionFailureCount;

    // Return code for this structure.
    // Applications must check BmapiGetISCSIRuntimeStatistics() to be BMAPI_OK
    // before using 'return_code' from each individual structure.
    U32 return_code;

} BM_ISCSI_STATS;



#define BM_ISCSI_SESSION_STATS_VER        1

typedef struct
{
    //  Version is defined as BM_ISCSI_SESSION_STATS_VER.
    //  'version' is required upon input
    U32 version;

    // Unique Adapter Id
    // Must be filled on input. It is available from 'BM_ISCSI_STATS'.
    U64 UniqueAdapterId;

#define BMAPI_ISCSI_STATS_SESSION_ID        0x00000001  // Request to fill SessionId
#define BMAPI_ISCSI_STATS_SESSION_NAME_LEN  0x00000002  // Request to fill Session Name
Length
#define BMAPI_ISCSI_STATS_SESSION_NAME      0x00000004  // Request to fill Session Name
#define BMAPI_ISCSI_STATS_SESSION_STATS     0x00000008  // Request to fill Session Stats
#define BMAPI_ISCSI_STATS_SESSION_ID_DONE   0x00000010  // Session ID is retrieved
    U32 flags;

    // Unique Session Id
    // If 'BMAPI_ISCSI_STATS_SESSION_ID' is not set on input, BMAPI expects
    // caller will provide the 'SessionId' as input and BMAPI will use
    // 'SessionId' to find and collect information.
    U64 SessionId;

    // Unique Session Name
    U8 *pSessionName;

    // Unique Session Name Length
    U32 SessionNameLength;

    //////////////////////////////
    //////Session Statistics//////
    //////////////////////////////
    // Number of bytes sent over this session
    U64 BytesSent;

    // Number of bytes received over this session
    U64 BytesReceived;

    // Number of PDU sent over this session
    U64 PDUCommandsSent;

    // Number of PDU received over this session
    U64 PDUResponsesReceived;

    // Count of Number of Digest errors occured in this session
    U64 DigestErrors;

    // Count of Number of ConnectionTimeout errors occured in this session
    U64 ConnectionTimeoutErrors;

    // Count of Number of Format errors occured in this session
    U64 FormatErrors;
    // Return code for this structure.
    // Applications must check BmapiGetISCSISessionStatistics() to be BMAPI_OK
    // before using 'return_code' from each individual structure.
    U32 return_code;

} BM_ISCSI_SESSION_STATS;



// Management processors definition for BmapiGetMgmtProcessors()
#define BMAPI_MGMT_PROC_NONE        0           // no management FW support
#define BMAPI_MGMT_PROC_RXCPU       0x00000001
#define BMAPI_MGMT_PROC_TXCPU       0x00000002
#define BMAPI_MGMT_PROC_APE         0x00000004



//
// BM_TCP_OFLD_STATS is available only for NDIS 5.2 or later.
// It also depends on the NDIS miniport's chimney offload capabilities.
//
#define BM_TCP_OFLD_STATS_VER   1

typedef struct
{
    //  Version is defined as BM_TCP_OFLD_STATS_VER.
    //  'version' is required upon input
    U32 version;

    // Return code for this structure.
    // Applications must check BmapiOfldStats() to be BMAPI_OK
    // before using 'return_code' from each individual structure.
    U32 return_code;

    U64 InSegments;
    U64 OutSegments;
    U32 CurrentlyEstablished;
    U32 ResetEstablished;
    U32 RetransmittedSegments;
    U32 InErrors;
    U32 OutResets;

} BM_TCP_OFLD_STATS;



//
// BM_IP_OFLD_STATS is available only for NDIS 5.2 or later.
// It also depends on the NDIS miniport's chimney offload capabilities.
//
#define BM_IP_OFLD_STATS_VER    1

typedef struct
{
    //  Version is defined as BM_IP_OFLD_STATS_VER.
    //  'version' is required upon input
    U32 version;

    // Return code for this structure.
    // Applications must check BmapiOfldStats() to be BMAPI_OK
    // before using 'return_code' from each individual structure.
    U32 return_code;

    U64 InReceives;
    U64 InOctets;
    U64 InDelivers;
    U64 OutRequests;
    U64 OutOctets;
    U32 InHeaderErrors;
    U32 InTruncatedPackets;
    U32 InDiscards;
    U32 OutDiscards;
    U32 OutNoRoutes;

} BM_IP_OFLD_STATS;
```

```
//
// LLDP parameters.
//
#define BMAPI_DCBX_CONFIG_INV_VALUE     (0xFFFFFFFF)

enum
{
    BMAPI_LLDP_OVERWRITE_SETTINGS_DISABLE  = 0,
    BMAPI_LLDP_OVERWRITE_SETTINGS_ENABLE   = 1,
    BMAPI_LLDP_OVERWRITE_SETTINGS_INVALID  = BMAPI_DCBX_CONFIG_INV_VALUE
};

typedef struct
{
    // Indicates if to use the default NVRAM LLDP settings or to overwrite
    // them by other configurable settings. Default = No.
    U32 overwrite_settings;

    // Defines txTTL = MIN(msg_tx_hold * msg_tx_interva, 65535). Default = 4.
    U32 msg_tx_hold;

    // Timer interval in seconds between transmissions in fast transmission
    // mode. Default = 1.
    U32 msg_fast_tx;

    // The maximum of LLDPDU that can be transmitted consecutively.
    // Default = 5
    U32 tx_credit_max;

    // Timer interval in seconds between transmissions. Default = 30.
    U32 msg_tx_interval;

    // Determines how many intervals shall take place during fast transmission
    // mode. Default = 4
    U32 tx_fast;

    U32 reserve[20];

} BM_LLDP_CFG_PARAMS;



#define BM_LLDP_PARAMS_VER  2

typedef struct
{
    //  Version is defined as BM_LLDP_PARAMS_VER.
    //  'version' is required upon input
    U32 version;

    BM_LLDP_CFG_PARAMS  cfg;

    // Enumeration defined as following.
#define BMAPI_LLDP_TX_ONLY  0x01
#define BMAPI_LLDP_RX_ONLY  0x02
#define BMAPI_LLDP_TX_RX    0x03
#define BMAPI_LLDP_DISABLED 0x04
    U32 admin_status;

    // The chassis ID advertised by the peer.
    U32 remote_chassis_id[65];

    // The port ID advertised by the peer.
    U32 remote_port_id[65];

    // The chassis ID advertised by the local machine.
    // port MAC address
    U32 local_chassis_id[2];

    // The port ID advertised by the local machine.
    // function MAC address
    U32 local_port_id[2];
```

```
} BM_LLDP_PARAMS;



//
// DCBX protocol parameters.
//

// Local machine priority to application assignment.
// Traffic type enumeration: 0 -EtherType, 1 - port over TCP,
//                           2 - port over UDP, 3 - port over both.
// Default (two valid entries) =
// Priority=3, traffic type=0, app_id=0x8906 (FCoE)
// Priority=4, traffic type=1, app_id=3260 (iSCSI)
typedef struct
{
    U32 valid;

#define BMAPI_DCBX_INVALID_TRAFFIC_TYPE_PRIORITY    (0xFFFFFFFF)
    U32 priority;

#define BMAPI_DCBX_TRAFFIC_TYPE_ETH     0
#define BMAPI_DCBX_TRAFFIC_TYPE_PORT    1
    U32 traffic_type;

    U32 app_id;

    U32 reserve[8];

} BM_DCBX_ADMIN_PRIORITY_APP_TABLE;



typedef struct
{
    // Indicates if to enable DCB. When Enable DCB = enable and
    // admin_dcbx_enable = disable, static configuration is done.
    // This parameter is used only by the driver.
    U32 dcb_enable;

    // Indicates if to enable DCBX negotiation. Default = enable.
    U32 admin_dcbx_enable;

    // Only if the entire DCBX registry set is present and differ from
    // 0xFFFFFFFF (invalid value) the DCBX parameters are taken, otherwise
    // the data are ignored. (Except "admin_dcbx_enable" and "dcb_enable")

    // Indicates if to use the default NVRAM DCBX settings or to overwrite
    // them by other configurable settings. Default = No.
    U32 overwrite_settings;

    // Indicates if to perform negotiation according to CEE or IEEE.
    // Default = CEE.
#define BMAPI_DCBX_ADMIN_DCBX_VERSION_CEE   0
#define BMAPI_DCBX_ADMIN_DCBX_VERSION_IEEE  1
    U32 admin_dcbx_version;

    // Indicates if to enable ETS. Default = enable.
    U32 admin_ets_enable;

    // Indicates if to enable PFC. Default = enable.
    U32 admin_pfc_enable;

    // Indicates if to transmit the TC Supported TLV by the local machine.
    // Default = enable.
    U32 admin_tc_supported_tx_enable;

    // Indicates if to transmit the ETS configuration TLV by the local
    // machine. Default = enable.
    U32 admin_ets_configuration_tx_enable;

    // Indicates if to transmit the ETS recommendation TLV by the local
    // machine. Default = disable.
    U32 admin_ets_recommendation_tx_enable;

    // Indicates if to transmit the PFC TLV by the local machine.
    // Default = enable.
    U32 admin_pfc_tx_enable;

    // Indicates if to transmit the application TLV by the local machine.
    // Default = enable.
    U32 admin_application_priority_tx_enable;

     // Indicates if the local machine is willing to accept the ETS
    // configuration recommended by the peer. Default = willing.
    U32 admin_ets_willing;

    // Indicates if the ETS recommendation sent from the local machine is
    // valid. Default = no.
    U32 admin_ets_reco_valid;

    // Indicates if the local machine is willing to accept PFC configuration
    // from the peer. Default = willing.
    U32 admin_pfc_willing;

    // Indicates if the local machine is willing to accept application
    // priority assignment configuration from the peer. Default = willing.
    U32 admin_app_priority_willing;

    // Local machine bandwidth percentage allocation. For example:
    // Default profile #0: [0,50,50,0,0,0,0,0]
    // Default profile #1 (for OEM1): [0,33,66,0,0,0,0,0]
    // Default profile #2 (for OEM2): [0,33,33,33,0,0,0,0]
    U32 admin_configuration_bw_percentage[8];

    // Local machine priority to priority group assignment.
    // Default profile #0: [0,1,0,2,1,0,0,0]
    // Default profile #1: [0,1,0,2,2,0,0,0]
    // Default profile #2: [0,1,0,2,3,0,0,0]
    U32 admin_configuration_ets_pg[8];

    // Recommendation to peer for bandwidth percentage allocation.
    // Default = zeros.
    U32 admin_recommendation_bw_percentage[8];

    // Recommendation to peer for priority to priority group assignment.
    // Default = zeros.
    U32 admin_recommendation_ets_pg[8];

    // Local machine per-priority PFC enabled/disabled (bitmap).
    // Default profile #0: 00010000b
    // Default profile #1: 00011000b
    // Default profile #2: 01011000b
    U32 admin_pfc_bitmap;

    // Local machine priority to application assignment.
    // Traffic type enumeration: 0 -EtherType, 1 - port over TCP,
    //                           2 - port over UDP, 3 - port over both.
    // Default (two valid entries) =
    // Priority=3, traffic type=0, app_id=0x8906 (FCoE)
    // Priority=4, traffic type=1, app_id=3260 (iSCSI)
    BM_DCBX_ADMIN_PRIORITY_APP_TABLE    admin_priority_app[4];

    // Local machine default priority for applications not defined by the
    // application protocol TLV. Default = 1.
    U32 admin_default_priority;

    U32 reserve[20];

} BM_DCBX_CFG_PARAMS;



#define BM_DCBX_PARAMS_VER  2

typedef struct
{
    //  Version is defined as BM_DCBX_PARAMS_VER.
    //  'version' is required upon input
    U32 version;

    BM_DCBX_CFG_PARAMS  cfg;
```

```
// The number of traffic classes that the local machine supports.
U32 local_tc_supported;

// The number of traffic classes that may simultaneously support
// PFC on the local machine.
U32 local_pfc_caps;

// The number of traffic classes that the peer supports.
U32 remote_tc_supported;

// The number of traffic classes that may simultaneously support
// PFC at the peer.
U32 remote_pfc_cap;

// Indicates if the peer is willing to accept the ETS configuration
// recommended by the local machine.
U32 remote_ets_willing;

// Indicates if the ETS recommendation sent from the peer is valid.
U32 remote_ets_reco_valid;

// Indicates if the peer is willing to accept PFC configuration.
U32 remote_pfc_willing;

// Indicates if the peer is willing to accept application priority
// assignment configuration.
U32 remote_app_priority_willing;

// Peer bandwidth percentage allocation configuration.
U32 remote_configuration_bw_percentage[8];

// Peer priority to priority group assignment configuration.
U32 remote_configuration_ets_pg[8];

// Peer bandwidth percentage allocation recommendation.
U32 remote_recommendation_bw_percentage[8];

// Peer priority to priority group assignment recommendation.
U32 remote_recommendation_ets_pg[8];

// Peer per-priority PFC enabled/disabled (bitmap).
U32 remote_pfc_bitmap;

// Peer priority to application assignment.
BM_DCBX_ADMIN_PRIORITY_APP_TABLE remote_priority_app[16];

// Indicates if ETS is enabled.
U32 local_ets_enable;

// Indicates if PFC is enabled.
U32 local_pfc_enable;

// Operational bandwidth percentage allocation
U32 local_configuration_bw_percentage[8];

// Operational priority to priority group assignment.
U32 local_configuration_ets_pg[8];

// Operational per-priority PFC enabled/disabled (bitmap).
U32 local_pfc_bitmap;

// Operational priority to application assignment.
BM_DCBX_ADMIN_PRIORITY_APP_TABLE local_priority_app[16];

// When set indicates that there was a mis-match between the local
// machine and the peer PFC configuration.
U32 pfc_mismatch;

// When set indicates that there was a mis-match between the local
// machine and the peer priority to application assignment configuration.
U32 priority_app_mismatch;

// Statistics.
U32 dcbx_frames_sent;
U32 dcbx_frames_received;
```

```
    U64 pfc_frames_sent;
    U64 pfc_frames_received;

} BM_DCBX_PARAMS;



#define BM_MBA_PARAMS_VER  1

typedef struct
{
    //  Version is defined as BM_MBA_PARAMS_VER.
    //  'version' is a required input field.
    U32 version;

    // boolean value
    // not applicable if no MBA FW in NVRAM
    // not applicable for SW or HW selfboot
    U32 mba_enable;

#define BMAPI_MBA_BOOT_PROT_NONE    0
#define BMAPI_MBA_BOOT_PROT_PXE     1   // default
#define BMAPI_MBA_BOOT_PROT_RPL     2
#define BMAPI_MBA_BOOT_PROT_BOOTP   3
#define BMAPI_MBA_BOOT_PROT_ISCSI   4
#define  BMAPI_MBA_BOOT_PROT_FCOE      5    //  Only  for  57710  type  of  devices  (57712  and
later)
    U32 boot_prot;

#define BMAPI_MBA_BOOTSTRAP_AUTO    0   // default
#define BMAPI_MBA_BOOTSTRAP_BBS     1
#define BMAPI_MBA_BOOTSTRAP_INT18H  2
#define BMAPI_MBA_BOOTSTRAP_INT19H  3
    U32 bootstrap;

    // boolean value
    U32 banner_enable;

    // in seconds (0-15)
    U32 banner_timeout;

    // false for Ctrl-S, true for Ctrl-B
    U32 hot_key;

#define BMAPI_MBA_SPEED_AUTO        0   // default
#define BMAPI_MBA_SPEED_10H         1
#define BMAPI_MBA_SPEED_10F         2
#define BMAPI_MBA_SPEED_100H        3
#define BMAPI_MBA_SPEED_100F        4
#define BMAPI_MBA_SPEED_1000H       5
#define BMAPI_MBA_SPEED_1000F       6
    // For BMAPI_BRCM57710, only BMAPI_MBA_SPEED_AUTO is applicable.
    U32 link_speed;

    // boolean value
    U32 wol_enable;

    // boolean value
    U32 vlan_enable;

    // 0..1023
    U32 vlan_id;

    // number of retry
    // applicable only to NXII
    U32 retry;

} BM_MBA_PARAMS;




#pragma pack(pop)
```