

INTO THE new  
**BREW 2007 CONFERENCE**

# Custom OEM Acceptance Test (OAT) Modules

Mahesh "Hesh" Rohera, Engineer  
QUALCOMM Incorporated





## Custom OAT Modules

- Why do I need Custom OAT Modules?
- Overview of OAT Architecture
- Write your own custom OAT Module
- OAT Framework and Logging APIs
- Integrate your custom module in PEK Studio
- References



## Why Do I need Custom OAT Modules?

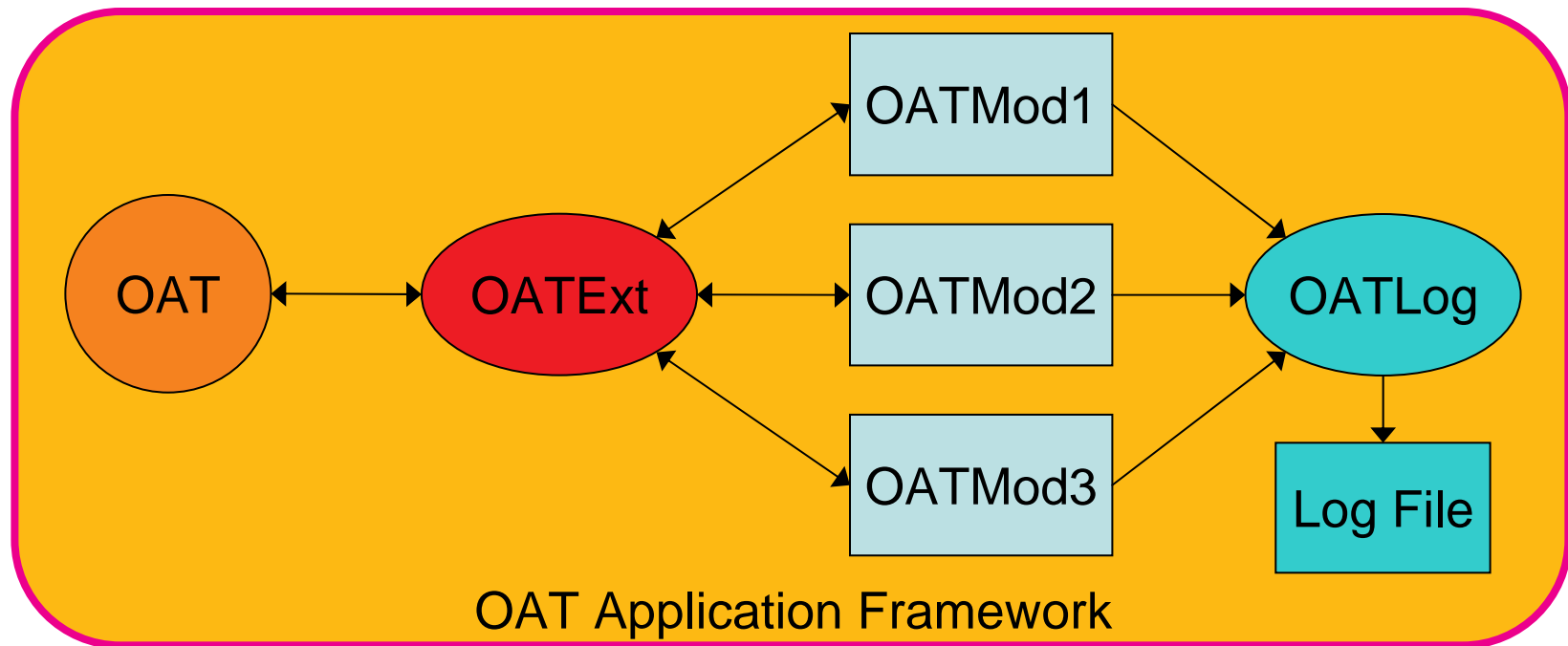
- Standard OAT modules
  - Test standard BREW-defined APIs<sup>®</sup>
    - > Behavior may be influenced by OEM-layer implementation
  - Same test cases run on all BREW<sup>®</sup> handsets to ensure cross-platform conformance
- Custom OAT modules
  - Test custom-developed BREW-style Extensions
  - Should be run on all platforms that expose the API
  - Should be run on all shipped builds as a regression test of the interface



## Why Do I need Custom OAT Modules?

- GOAL: Binary compatibility between BREW applications and all handsets exposing the API.

# OAT Architecture



- OAT is a BREW Application
- OAT modules are BREW Extensions
- OAT loads and runs one module at a time



## OAT Architecture

- OAT framework defines two classes of test cases
  - “Automated”
  - “Interactive”
- Users choose the running mode for OAT
  - All
  - Automated
  - Interactive
  - Configured



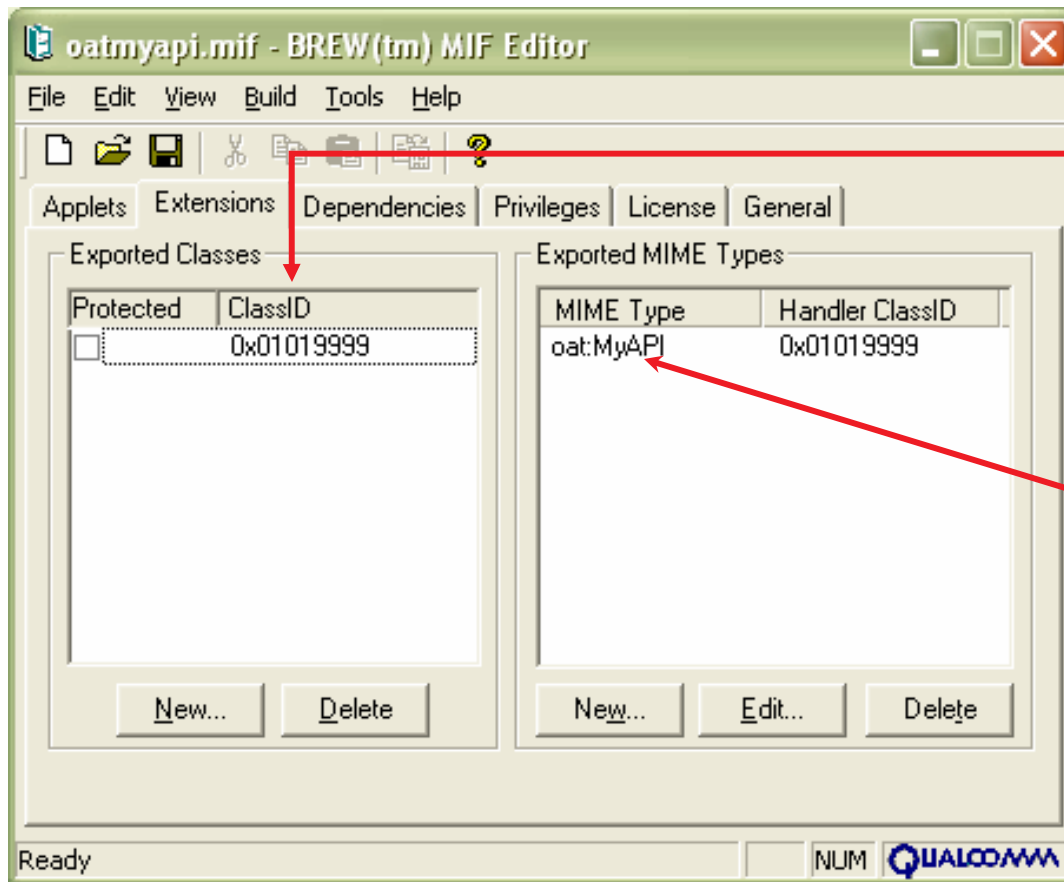
# OAT Architecture

## Test Case Results

- OAT\_SUCCESS
- OAT\_FAILED
- OAT\_VERIFY
- OAT\_UNDETERMINED
- OAT\_UNSUPPORTED
- OAT\_POSTPROCESS

# Writing a new OAT Extension

## OAT Module MIF File



1. Get a Class ID from Class ID generator

2. Export the Class ID

3. Export MIME Type (Base Class AEECLSID\_APP)

4. Put the MIF file in the "pek\oat\src\mif" directory



## Writing a new OAT Extension

- Define a data structure for your OAT test's private member variables

```
typedef struct _OATMyAPI
{
    OAText e; // OAText is always first

    // Class member variables can follow
    int nTestCount;
}OATMyAPI;
```

# Writing a new OAT Extension

- Write the `_CreateInstance()` function
  - Use `OATEXT_New(Ex)` to create the module

```
boolean OATEXT_NewEx(  
    int nSize,          // sizeof(OATMyAPI)  
    AEECLSID cls,      // Pass directly from CreateInstance  
    IShell * ps,       // Pass directly from CreateInstance  
    IModule * pMod,    // Pass directly from CreateInstance  
    void **ppMod,      // Pass directly from CreateInstance  
    PFNSTARTTEST pfnst, // StartTest() function  
    PFNENDTEST pfnend, // EndTest() function  
    PFNHANDLEEVENT pfnEvtHandler) // HandleEvent() fn
```

- NOTE: `OATEXT_New()` is same as `OATEXT_NewEx()` except it doesn't take `pfnEvtHandler`.



## Writing a new OAT Extension

- Write the StartTest() function

```
TestNodeList * OATMyAPICls_StartTest (  
    IOATExt * pExt,  
    char * name)
```

- StartTest() should MALLOC, populate and return a TestNodeList containing all the test cases in the OAT module
  - > TIP: Use OATTestXXXX() macros from OATExt.h to easily populate the fields in a TestNode.
- Copy the Module Name to the name parameter before returning



## Writing a new OAT Extension

- Write the EndTest() function

```
void OATMyAPICls_EndTest(IOATExt * pExt)
```

- OAT framework will execute EndTest() after all test cases have been run
- EndTest() should clean up anything that needs to be cleaned up after the test run
  - > Free up memory that the test cases MALLOC'd while running
  - > Release any interfaces that were created
- NOTE: TestNodeList and TestNodes are FREE'd by the framework



## Writing a new OAT Extension

- Write a HandleEvent() function (Optional)

```
boolean OATMyAPI_HandleEvent(IOATExt *pOATExt,  
                             AEEEvent evt,  
                             uint16 wParam,  
                             uint32 dwParam)
```

- HandleEvent() function is only required for OAT Extensions that need to process BREW events (EVT\_NOTIFY for example)
- OAT Framework will forward BREW events to OATMyAPI\_HandleEvent()



# Writing a new OAT Extension

## Declaring OAT test cases

- OATExt.h provides macros for easily declaring OAT test cases
- OAT framework will run Interactive test cases before executing the Automated test cases

## Automated Test Cases    Interactive Test Cases

- |                     |                              |
|---------------------|------------------------------|
| – OATTestAuto       | – OATTestInt                 |
| – OATTestAlways     | – OATTestIntEx               |
| – OATTestAlwaysAuto | – OATTestIntAsync            |
| – OATTestAsync      | – OATTestIntSel              |
|                     | – OATTestIntAsyncEx          |
|                     | – OATTestStopStartBackground |



## OATTestAuto

- Used for test cases that can execute entirely within the test function
- OAT framework will proceed to next test case when the test function returns

```
OATTestAuto(tn->test[tc], // TestNode  
            OATMyAPI_TestAuto, // Test function  
            "MyAPI_TestAuto") // Test Name
```



## OATTestAlways

- Tests “Always” run irrespective of user configuration
- Useful in executing pre-condition test cases
- OAT Module must call IOATEXT\_ResumeTest in order for OAT to proceed to the next test

```
OATTestAlways(tn->test[tc], // Test Node
              OATMyAPI_TestAlways, // Test Fn
              "MyAPI_TestAlways" ) // Test Name
```



## OATTestAlwaysAuto

- Tests “Always” run whenever user runs any Automated test cases
- Useful in ensuring that pre-condition test cases run before certain Automated tests
- OAT Module must call IOATEXT\_ResumeTest in order for OAT to proceed to the next test

```
OATTestAlwaysAuto(tn->test[tc], // TestNode  
    OATMyAPI_TestAlwaysAuto, // Test Fn  
    "MyAPI_TestAlwaysAuto") // Test Name
```

## OATTestAsync

- Used for automated test cases that need to process callbacks, timers or events before making a Pass/Fail decision
- OAT framework will proceed to next test case when OAT module calls IOATEXT\_ResumeTest()

```
OATTestAsync(tn->test[tc], // Test Node
             OATMyAPI_TestAsync, // Test Fn
             "MyAPI_TestAsync" ) // Test Name
```

## OATTestInt

- Requires the user to Pass or Fail the test
- Pass/Fail/Retry menu is provided to user when the test function returns
- EVT\_OAT\_RESULT is sent to OAT Module to notify the user's Pass/Fail decision
  - Or the OAT module can ignore this Event and OAT Framework will log the decision.

```
OATTestInt(tn->test[tc], // TestNode
           OATMyAPI_TestInt, // Test Fn
           "MyAPI_TestInt") // Test Name
```

# OATTestInt



User is presented with Pass/Fail/Retry decision when test Function returns

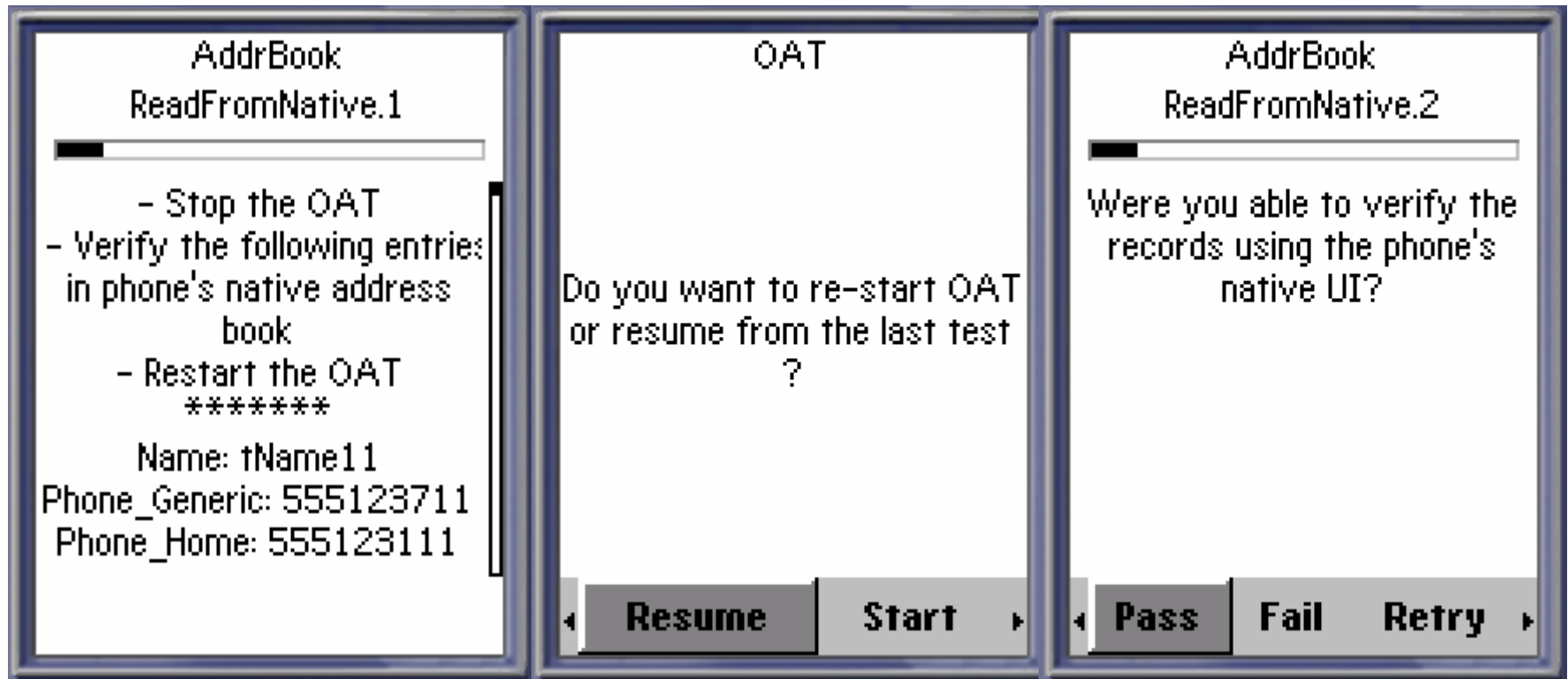


## OATTestIntEx

- Requires the user to end OAT momentarily (Ex. to check the native Address Book)
- Test state is saved in OAT Framework
- When OAT resumes user is presented with Pass/Fail/Retry decision
- User's decision is delivered to OAT Module with EVT\_OAT\_RESULT

```
OATTestIntEx(tn->test[tc], // Test Node
             OATMyAPI_TestIntEx, // Test Fn
             OATMyAPI_TestIntEx_Resume, // Resume Fn
             "MyAPI_TestIntEx") // Test Name
```

# OATTestIntEx



1. Ask the user to exit OAT and do something

2. User is asked to resume the test on next launch

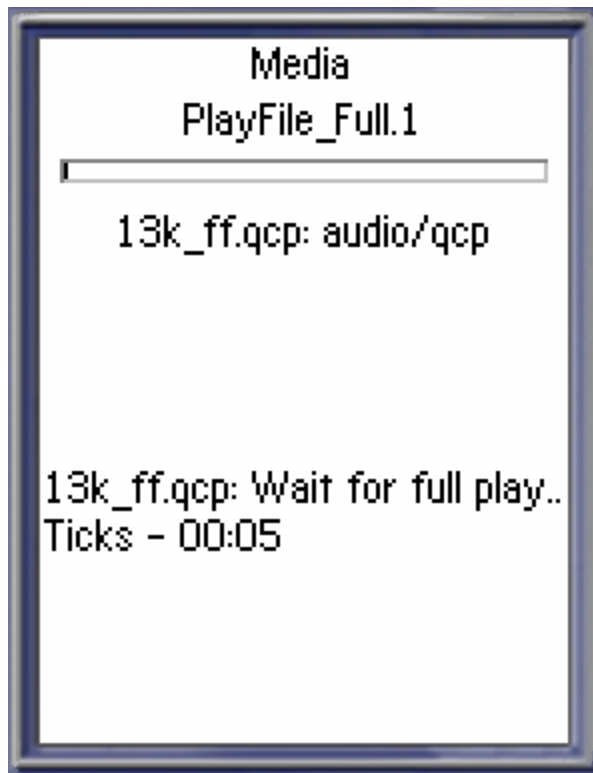
3. Ask the user a Pass/Fail question

## OATTestIntAsync

- Requires the user to Pass or Fail the test
- Pass/Fail/Retry menu is delayed until OAT calls IOATEXT\_ResumeTest()
  - This allows the OAT Module to process events or receive callbacks before asking the user for Pass/Fail.
- User's decision is delivered to OAT Module with EVT\_OAT\_RESULT

```
OATTestIntAsync (tn->test[tc], //Test Node
OATMyAPI_TestIntAsync, // Test Fn
"MyAPI_TestIntAsync") // Test Name
```

# OATTestIntAsync



1. Test case starts and processes callbacks/events



2. Call `IOATEXT_ResumeTest()` to present user with Pass/Fail/Retry decision



## OATTestIntSel

- Requires the user to Pass or Fail the test
- Pass/Fail/Retry menu is delayed until the user presses the SELECT key
  - This allows the OAT module to use the entire display for the test case
- User's decision is delivered to OAT Module with EVT\_OAT\_RESULT

```
OATTestIntSel (tn->test[tc], // Test Node
OATMyAPI_TestIntSel,      // Test Fn
"MyAPI_TestIntSel")      // Test Name
```

# OATTestIntSel



1. Test case starts and may consume the entire display



2. User is presented with Pass/Fail/Retry decision after pressing SELECT

## OATTestAsyncEx

- Automatic Pass/Fail decision
- User input might be required in order for the test to progress (to answer a privacy dialog, for example)
  - Therefore, it is considered as an interactive test case.

```
OATTestAsyncEx (tn->test[tc], // Test Node
OATMyAPI_TestAsyncEx,      // Test Fn
"MyAPI_TestAsyncEx" )      // Test Name
```



## OATTestStopStartBackground

- Used to send OAT to the background momentarily (Ex. To ask the user to perform an action)
- Module must call IOATEXT\_GoBackground()
- When OAT resumes automatic Pass/Fail decision is made

```
OATTestStopStartBackground(  
    tn->test[tc],                // Test Node  
    OATMyAPI_TestSSBg,          // Test Fn  
    OATMyAPI_TestSSBg_Resume    // Test Resume Fn  
    "MyAPI_TestSSBg")           // Test Name
```



# OAT Logging Facilities

## IOATLOG (see OATLog.h)

- IOATLOG\_WriteModVer
  - > Writes the version string to the OAT Log
  - > **Ex. OAT AddrBook Ver:3.1.5.33**
- IOATLOG\_Start(Ex)
  - > Use this at the beginning of the test function to log start of test
  - > **Ex. NAME:MyAPI\_TestIntAsync.1:INT:TestIntAsync**
- IOATLOG\_Status(V)
  - > Logs the result of a test case and optional message
  - > **Ex. STATUS:SUCCESS: Yay. The test passed.**



# OAT Logging Facilities

## IOATLOG

- IOATLOG\_Msg(V)
  - > Logs an explanatory message
  - > **Ex. MSGH: Received first expected callback**
  
- IOATLOG\_MsgPri(V)
  - > Logs an explanatory message with priority level
  - > **Ex. MSGM:Failed to open media file**



# OAT Logging Facilities

## IOATLOG

- (New helpers included in PEK 3.1.5FP02)
  - IOATLOG\_WriteString
  - IOATLOG\_WriteInt
  - IOATLOG\_WriteBoolean
  - IOATLOG\_WriteBin
  - IOATLOG\_WriteStruct
    - > Logs Name-Value pairs
    - > Useful when “post processing” of OAT results is required
    - > **Ex. MyStruct:{1234, TRUE}**
    - > **Ex. CFGI\_THEMENAME:fs:/mod/19917/t1/theme.bar**

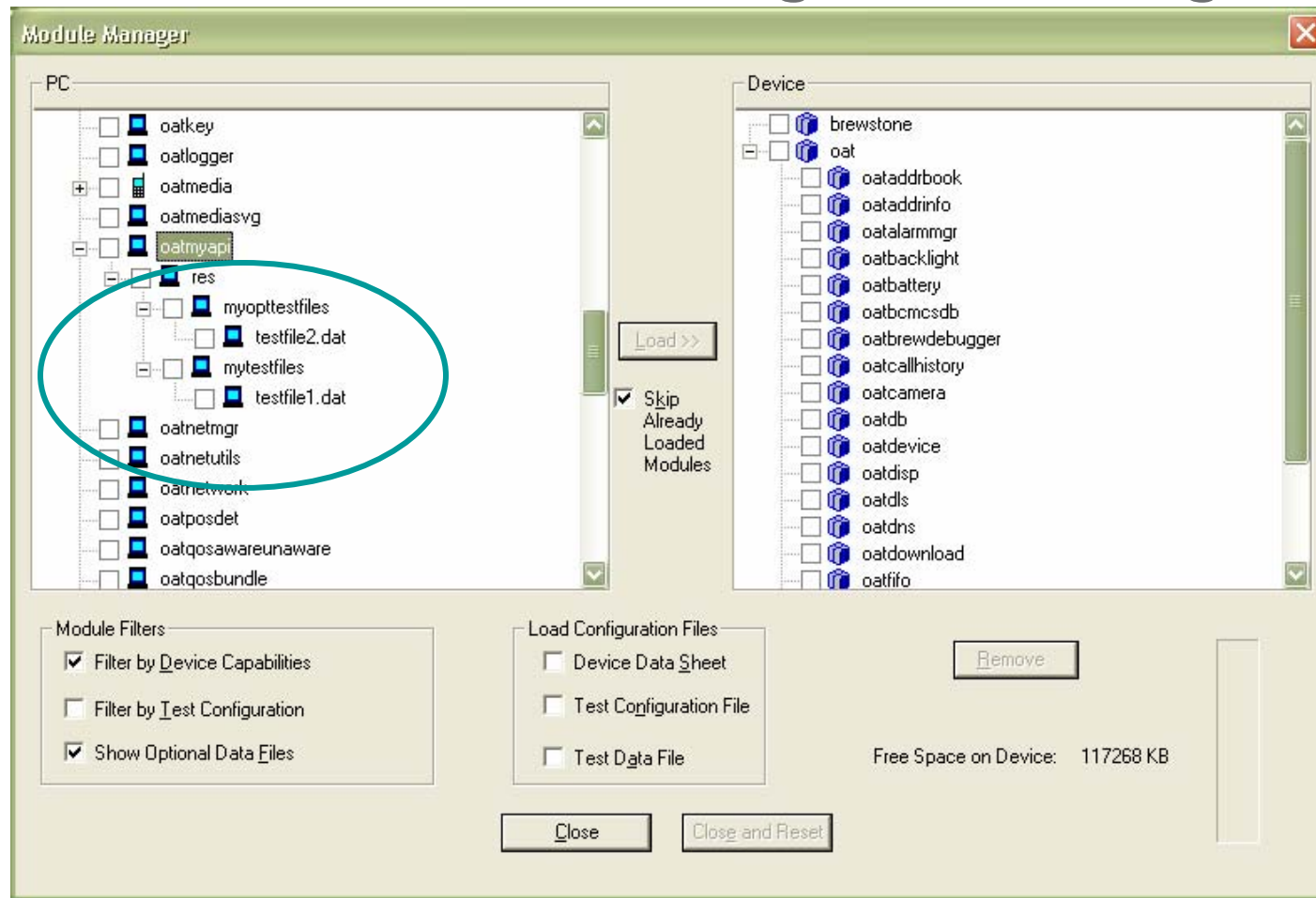


## Integrating with PEK Studio

- PEK Studio automatically detects all modules in the “pek\oat\bin” directory
- TIPS
  - Create your OAT module directory in the pek\oat\src directory. Ex. pek\oat\src\oatmyapi
  - Put the MIF File in pek\oat\src\mif directory
  - Construct a makefile for your OAT module, using existing modules as examples
  - This makefile builds a .MOD file and copies it to pek\oat\bin directory

# Integrating with PEK Studio

“Modules->Module Manager...” dialog





## Integrating with PEK Studio

PEK Studio “Configure -> OAT Tests...”

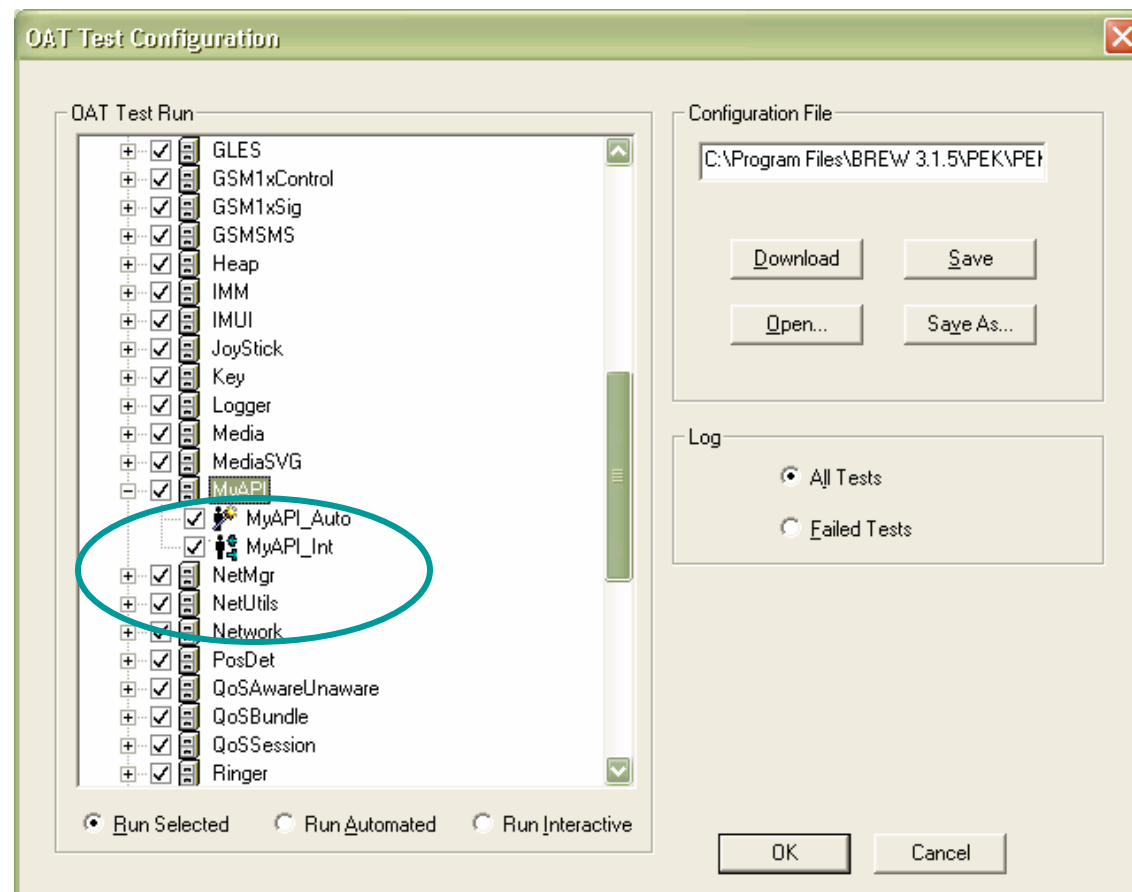
- Easier beginning in PEK 3.1.4
- Implemented by reading in testdesc.txt from each module’s directory

```
/*  
  { {TEST_LIST(MyAPI)  
    MyAPI_Auto:auto  
    MyAPI_Int:int  
  } }TEST_LIST(MyAPI)  
*/
```

- Format is <TestName>:<int/auto>

# Integrating with PEK Studio

PEK Studio "Configure -> OAT Tests" dialog





# Integrating with PEK Studio

## Adding Files to the Device File System

- Useful if your OAT module requires files be placed on the device file system
- How?
  - > Beginning in PEK 3.1.5, each module has PEKModuleData.xml file to describe additional assets to be loaded to the device

# Integrating with PEK Studio

## PEKModuleData.xml file

```
<PEKITEMS>
  <OATMODULE NAME="oatmyapi">
    <DEVICE_CLASSIDS>
      <CLASSID NAME="IDS_AEECLSID_LAST">
        <DATAFILES DST="$BREW_ROOT_DIR/oat/myapi"
          SRC="$PEK_ROOT\oat\src\oatmyapi\res\mytestfiles"
          TYPE="D2D" PARENT_DIRS="1"
          MANDATORY="1"></DATAFILES>
        <DATAFILES DST="$BREW_ROOT_DIR/oat/myapi"
          SRC="$PEK_ROOT\oat\src\oatmyapi\res\myopttestfiles"
          TYPE="D2D" PARENT_DIRS="1"
          MANDATORY="0"></DATAFILES>
      </CLASSID>
    </DEVICE_CLASSIDS>
  </OATMODULE>
</PEKITEMS>
```



# Custom OAT Modules

- References
  - BREW Porting Evaluation Kit (PEK) 3.1.5 User Guide Appendix B: Creating Custom OAT Modules
  - Consider existing OAT modules as examples



Questions?



Thank You