```
#ifndef ITEST_H
#define ITEST_H 1


/**
* @brief
*     The interface for all Test objects.
*
* All test types will provide default implementation of some virtual methods:
* init(), preExec(), execute() and postExec(). These methods become the test
* engineer's entry points for overriding default behavior and setting any test-
* specific parameters.
*/
class ITest: public IDUTFlowable
{
        // datadef
public:
        /*
         * @brief
         *      array of PatternTree
         */
        typedef OFCArray<PatternTree *> PatternTreeArray_t;
        /*
         * @brief
         *  array of TestCondition
         */
        typedef OFCArray<TestCondition *> TestConditionArray_t;


    /**
     * @brief
     *      enumeration for test result
     * @remarks
     *      The following status can be returned and their meanings are
     *          PASS            : executed and the result was pass.
     *          FAIL            : executed and the reuslt was fail.
     *          FAILED_TO_RUN   : got error(s) during execution.
     *          NOT_TESTED      : not executed yet.
         *          OTHER           : non-standard test-specific status.
     */
    enum TestStatusEnum
    {
        PASS = 1,
        FAIL,
        FAILED_TO_RUN,
        NOT_TESTED,
                OTHER
    };

public:

    /**
     * @brief
     *      Set a name for this Test object
     * @param name [in]:
     *      Name for this test object
     */
    virtual void setName( const OFCString& name ) = 0;

    /**
     * @brief
     *      Get a name of this Test object
     * @return
         *      Name of this test object
     */
```

```
    virtual const OFCString& getName( void ) const = 0;

        /**
     * @brief
     *      Get a name of test class which implements ITest
     * @return
     *      Name of test class
     */
    virtual const OFCString& getTestClassName( void ) const = 0;

    /**
     * @brief
     *      Set path of a library where test was loaded from.
     * @param path[out]
     *      Path of the library.
     */
    virtual void setLibraryPath( const OFCString& path ) = 0;

    /**
     * @brief
     *      Get path of a library where test was loaded from.
     * @return
     *      Path of the library.
     */
    virtual const OFCString& getLibraryPath() const = 0;

    /**
         * @brief
         *      Specifies the test id.
         *
         * @param testId
         *      The test Id.
         */
        virtual void setId(int testId) = 0;

        /**
         * @brief
         *      Retrieves the test id.
         *
         * @return
         *      The test Id.
         */
        virtual int getId() const = 0;

    /**
         * @brief
         *      Specifies the description for the test.
         *
         * @param desp
         *      The description text.
         */
        virtual void setDescription(const OFCString &desp) = 0;

        /**
         * @brief
         *      Retrieves the test description.
         *
         * @return
         *      The test description.
         */
        virtual const OFCString getDescription() const = 0;

        /**
         * @brief
         *      Provides a complete list of any "OTHER" sub-status values.
```

```
         *
         * @param statues
         *     An array of all "OTHER" sub-status values.
         *
         * @remarks
         *     If a test class only support standard pass-fail status, it
         *     will return a empty array.
         *
         *     GUIs can retrieve this list and allow users to set any
         *     colors (or any others( appropriately before testing occurs.
         */
        virtual void getAllOtherSubstatuses(OFCStringArray_t &statuses) const = 0;

        /**
     * @brief
     *      Gets Pass/Fail status.
     * @return
     *      Pass/Fail stauts as the result of the test execution.
         * @remarks
         *      If the returned status is OTHER, getOtherSubStatus() should be
         *      called to get detailed specific fined-granule status.
     */
    virtual TestStatusEnum getPassFailStatus() const = 0;

    /**
     * @brief
     *      Gets Pass/Fail status of specific DUT.
     * @param   dutID
     *      The dutID which gets result.
     * @return
     *      Pass/Fail stauts as the result of the test execution
         * @remarks
         *      If the returned status is OTHER, getOtherSubStatus() should be
         *      called to get detailed specific fined-granule status.
     */
    virtual TestStatusEnum getPassFailStatus(const DUTID_t& dutID) const = 0;

        /**
         * @brief
         *      Gets fined non-standard other sub-status of specific DUT.
         * @param dutID
         *      The ID of the DUT which gets the status.
         * @return
         *      The fined other sub-status string.
         * @remarks
         *      Only getPassFailStatus return OTHER, this function will be called.
         *      Otherwise, an exception will be raised.
         */
        virtual const OFCString getOtherSubstatus(DUTID_t dutID) const = 0;

        /**
         * @brief
         *     Initializes the test.
         */
        virtual void init() = 0;

    /**
     * @brief
     *      initialize test
     * @return
     *      boolean status for validation.
     * @remarks
     *      postInit() is to do user level initialization.
     *      After system initializes the test with init(),
     *      this method can be called by user's interaction.
```

```
     */
    virtual bool postInit() = 0;

        /**
         * @brief
         *     Specifies the pattern list tree.
         *
         * @param patternTree
         *     The pattern list tree.
         */
        virtual void addPatternList(PatternTree * const patternTree) = 0;

        /**
         * @brief
         *     Specifies the pattern list tree with index number.
         *
     * @param id [in]:
     *     Id number for this pattern tree when test will take
     *     multile PatternLists.
         * @param patternTree
         *     The pattern tree.
         */
        virtual void setPatternList(const int id,
                                    PatternTree * const patternTree) = 0;

    /**
     * @brief
     *      Get the pattern lists that the ITest derived object is associated with.
     * @param patLists [out]:
     *      array of pattern lists
     */
    virtual void getPatternLists( PatternTreeArray_t& patLists ) = 0;
        /**
         * @brief
         *              Get the number of pattern lists specified to the ITest derived
object
         * @return
         *              number of pattern lists
         */
        virtual size_t getNumOfPatternLists() = 0;

        /**
         * @brief
         *     Specifies the test condition.
         *
         * @param testCondition
         *     The test condition to be set.
         */
        virtual void addCondition(TestCondition * const testCondition) = 0;

        /**
         * @brief
         *     Specifies the test condition.
     *
     * @param id [in]:
     *     Id number for this test condition when test will take
     *     multile TestConditions.
         *
         * @param testCondition [in]:
         *     The test condition to be set.
         */
        virtual void setCondition(const int id,
                                  TestCondition * const testCondition) = 0;

    /**
```

```
     * @brief
     *      Get TestConditions that are associated with this ITest instance
     * @param testConds [out]:
     *      array of TestConditions
     */
    virtual void getConditions( TestConditionArray_t& testConds ) = 0;

        /**
         * @brief
         *              Get the number of TestConditions specified to the ITest derive
d object
         * @return
         *              number of TestConditions
         */
        virtual size_t getNumOfConditions() = 0;

    /**
     * @brief
     *    Specifies TestPlan which creates this test
     * @param pTPlan [in]:
     *    TestPlan which creates this test object
     */
    virtual void setTestPlan( ITestPlan * const pTPlan ) = 0;

    /**
     * @brief
     *    Get TestPlan which created this test object
     * @return
     *    ITestPlan pointer
     */
    virtual ITestPlan *getTestPlan(void) const = 0;

    /**
     * @brief
     *      Set debug mode to a test
     * @param mode [in]:
     *      true or false
     */
    virtual void setDebugMode( const int& mode ) = 0;

    /**
         * @brief Returns the message handler ID.
     */
        /// virtual unsigned long getMsgHandlerID() const = 0;
        /**
         * @brief Returns the GUID of corresponding proxy class
         */
        /// virtual const OFCString & getProxyGUID() const = 0 ;

    /**
         * @brief Gets the XML description for property page
         */
        /// virtual OFCString getXMLDescription() const = 0;

    /**
     * @brief Set/Get the parameters by string arguments
     */
    virtual OFCStatus setValues( const OFCStringArray_t& ) = 0;
    virtual void getValues( OFCStringArray_t& ) = 0;

        /** Gets the type of a parameter as a string.  If fieldName is
         *  empty, paramName is not a ParamGroup, and the type of the
         *  parameter is returned as a string.  If fieldName is nonempty,
         *  paramName is a ParamGroup, and fieldName is a field of that
         *  ParamGroup.
```

```
         *  @param paramName: The Parameter name.  It is a ParamGroup
         *     if fieldName is nonempty.
         *  @param fieldName: If nonempty, this is a field of a
         *     ParamGroup named paramName.
         *  @return: A string that is the type of the parameter or field
         *     of the ParamGroup.
         */
        virtual OFCString
        getType(const OFCString& paramName,
                        const OFCString& fieldName) const = 0;
    /**
     * @brief Updates the conditions for every bursts after modifying pattern lists
     */
    virtual OFCStatus updateBurstCondition() = 0;

    /**
         * @brief
         *      Get the IAnnotatable pointer associated with this object.
         *
         * @return IAnnotatable
         *     A pointer to the IAnnotatable associated with this
     *     instance.
         */
        virtual
    IAnnotatable*
    getIAnnotatable(void) = 0;

    ///\brief
    /// Modify the param values. This marks the IAnnotatable member dirty, and calls
    /// the setValues() method.
        virtual
    OFCStatus
    modifyValues(const OFCStringArray_t& ) = 0;

};

} // namespace OASIS


#endif // not ITEST_H
```