Protocol Solutions Group
3385 Scott Blvd. Santa Clara, CA 95054 Tel: +1/408.727.6600 Fax: +1/408.727.6622

InFusion™ API Reference Manual

For *InFusion*™ Software Version 1.00
Manual Version 1.01
April 4, 2005

Table of Contents

List of Figures

1. Introduction

1.1. General

The InFusion™ system is designed to provide impairment and monitoring capabilities over SAS and SATA protocols.

The system is composed of hardware (the InFusion™ device) and software (the InFusion™ SDK and the client applications). The InFusion™ device can be operated directly through its simple and intuitive front panel or through a client application running on a host machine.

The Infusion™ system can be used as a stand-alone tool or as part of a complex setup that may include one or more InFusion™ devices, several devices-under-test (DUTs), protocol analyzers (like the LeCroy's SASTracer™ and SATracer™) and other test & measurement tools.

The InFusion™ software package includes the LeCroy InFusion™ application that provides a comprehensive user-interface for controlling one and more InFusion™ systems. In addition to the application, the software consists of an API (Application Programming Interface) that allows 3rd-party implementations to access directly to the underlying InFusion™ SDK (Software Development Kit) functionalities. This is especially beneficial when the operation of the InFusion™ system is to be automated and integrated with other tools, without using the LeCroy application.

This reference manual describes the architecture and programming model of the InFusion™ SDK and details the API components.

1.2. Architecture

1.2.1. System Architecture

An InFusion™ system includes one or more InFusion™ devices that can be remotely controlled through an Ethernet connection.

To remotely control an InFusion™ device a host machine running the InFusion™ software sub-system is required. For now, the InFusion™ system supports only PCs that run Windows 2000 or Windows XP. Please refer to the User's Manual and ReadMe notes for additional information regarding the requirements for the host-machine. Figure 1-1 plots an architecture overview of the system.

The InFusion™ SDK is included in a single Windows DLL - the **INAPI.DLL**. Two additional DLLs, PSGLicensing.DLL and PSGUtil.DLL, are also required for running the system.

The application should load only the INAPI.DLL (dynamically or statically) to be able to use the InFusion™ functionalities.

A set of header files is included is installed as part of the API in the 'IncludeExp' directory. These files include all the definitions required for operating the InFusion™ SDK.

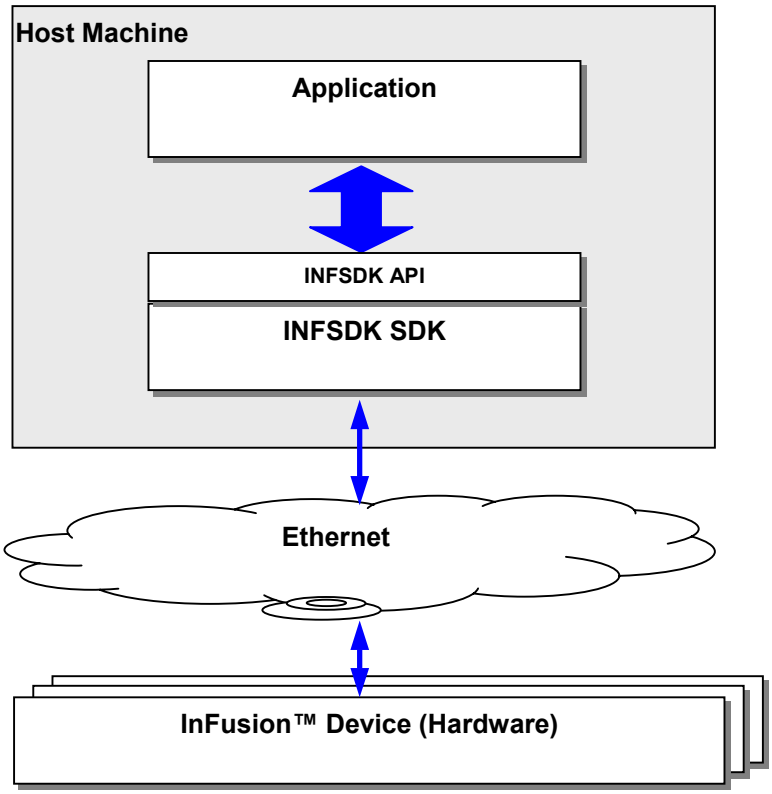


Figure 1-1 System Architecture

1.2.2. SDK Architecture

The InFusion™ API, the interface to the INF SDK, exposes only the functionalities that are essential for a client application to control the InFusion™ functionalities.

An object Oriented approach was used to organize the API into functional groups. A single object – the CINFAPI – is exported to the client application (see Figure 1-2).

The API object is composed out of three different objects, each representing a separate set of functionalities:

1. Monitor – Provides access to the monitoring, reporting and logging functionalities.
2. Device Manager – Provides access to control InFusion™ devices.
3. Configuration Manager– Provides accessibility to and control over to the configuration libraries.

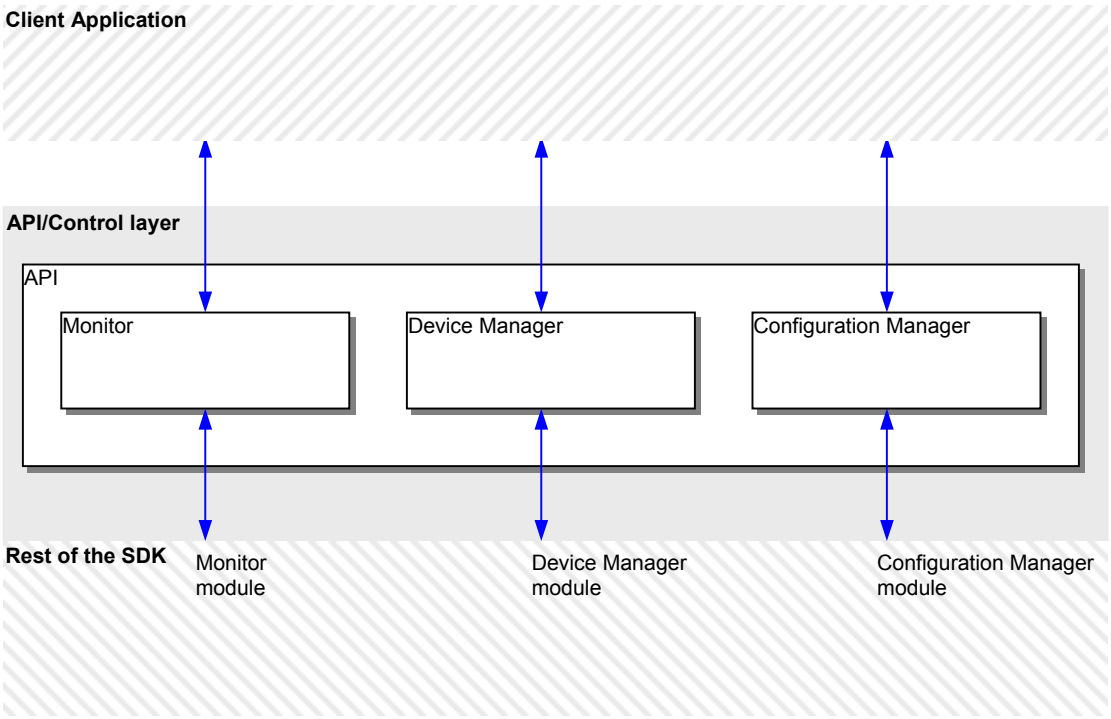


Figure 1-2 API Module Architecture

2. API

2.1. Installation

To install the API related directories and files, the API compoment has to be explicitly selected during installation.

2.1.1. Directories

After the installation is done the following set of directories should be located under the pre-determined InFusion install directory (see Figure 2-1):

- API – Root directory for the API files.
- APISamples – holds all the sample projects and sources fro the API.
- BinDebug, BinRelease – Hold the library files and DLLS for compilation and run-time of the applications.
- SDK\IncludeExp – Holds all the header files that include all the API definitions.

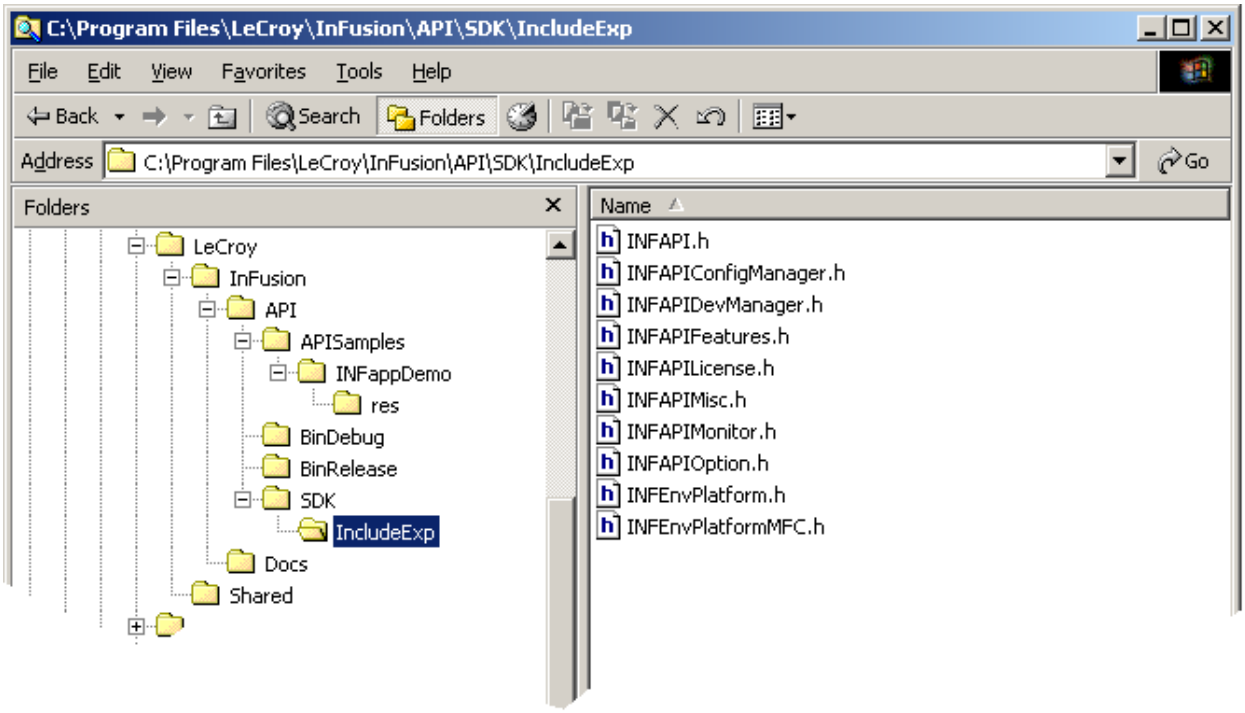


Figure 2-1 API Directories Layout

2.1.2. Header Files

The following header files are used for providing the definitions:

- INFAPI.h – Main header file. Includes all the rest of the API header files.
- INFAPIConfigManager.h – Definitions for the Configuration Manager functionality.
- INFAPIDevManager.h – Definitions for the Device Manager functionality.
- INFAPIFeatures.h – Definitions for the licensed features of the InFusion™ system.
- INFAPIMisc.h – Miscellaneous definitions (API return codes, for instance).
- INFAPIMonitor.h – Definitions for the Monitoring functionality.
- INFAPIOption.h – Definitions for API options.
- INFEnvPlatform.h – Definitions that relate to the software platform. For now, only Windows OS with Microsoft Foundation Class (MFC) is supported.
- INFEnvPlatformMFC.h – Definitions for MFC related classes.

For a regular client application, it should be enough to include the INFAPI.h file.

2.2. Definitions and structures

2.2.1. Naming Conventions

The three functionality groups are distinguished by different prefixes that are added to the names.

IB - marks the Device Manager functionality

IM - marks the Monitor functionality

IC - marks the Configuration Manager functionality

2.2.2. Return Codes

Whenever possible return codes of type EINFRC are returned from function calls.

EINFRC is based on the concept of HRESULT:

- A 2-bit code indicating severity, where zero represents success.
- A 2-bit reserved value.
- A 12-bit code indicating the module that returned the code.
- A 16-bit code describing the actual error or warning.

The definitions for the various return codes can be found in INFAPIMisc.h

2.2.3. Definitions Data structures

See description in relevant sections of the Device Manager, Monitor and Configuration Manager chapters.

2.3. API Functions

As described before only a single object of type CINFAPI is exposed to the client application.

All the SDK functionalities are accessible through the CINFAPI object or objects that are included in it.

2.3.1. Initialization and Termination

The client application must create an instance of the INFAPI object by using a ***new*** function.

After creating the object the object must be initialized by calling the `CINFAPI::Initialize()` function.

When the application is about to close, a call to `CINFAPI::Termiante()` is required to terminate the SDK functionalities.

Only after terminating the SDK functionality a ***delete*** function can be used to destroy the CINFAPI instance.

2.3.2. Access Functions

The INFAPI includes three main objects each representing one of the main SDK functionalities: CIBApiDeviceManager, CIMApiMonitor and CICApiConfigManager. The objects are created when the INFAPI is instantiated.

To access each one of the functionalities in these objects the `CINFAPI::GetDeviceManager()`, `CINFAPI::GetMonitor()`, and the `CINFAPI::GetConfigManager()` can be used respectively.

2.3.3. Functions List

Name	**Description**
`Initialize`	Initializes the CINFAPI object
`Terminate`	Terminates the operation of the CINFAPI object
`GetDeviceManager`	Returns a pointer to the Device Manager object
`GetMonitor`	Returns a pointer to the Monitor object
`GetConfigManager`	Returns a pointer to the Configuration manager object

2.4. API Notifications Callback

The API callback class, the CINFAPINotificationCallback , provides the client application a way to get notifications from the underlying InFusion™ SDK.
Once the application registers it's own callback fbction it would start getting notifications about changes to devices status and other system notifications.

Name	Description
`OnDeviceConnected`	Called when a connection to a device had been established.
`OnDeviceDisconnected`	Called when the connection to the device was terminated.
`OnEntityUpdated`	Called during and at the end of an entity (firmware or BusEngine) update process.
`OnScenarioUpdated`	Called during and at the end of an entity update.
`OnScenarioErased`	Called when a scenario was removed from the device.
`OnLicenseKeyUpdated`	Called during and at the end of an entity update process.
`OnSessionStarted`	Called when a session has started.
`OnSessionStopped`	Called when a session has stopped.
`OnDeviceJoined`	Called when detected that a new device connected to the network.
`OnDeviceLeft`	Called when detected that a device has disconnected from the network.
`OnDeviceBusy`	Called when detected that a device has connected to another host.
`OnDeviceReady`	Called when detected that a device is ready for connection.
`OnDeviceNameChanged`	Called when a device name has changed.
`OnDeviceHWStatusChanged`	Called when the hardwar4e status of a device has changed.
`OnDeviceDiscoveryStarted`	Called when a periodic discovery process has starting.
`OnDeviceDiscoveryStopped`	Called when the discovery process has stopped.

3. Device Manager

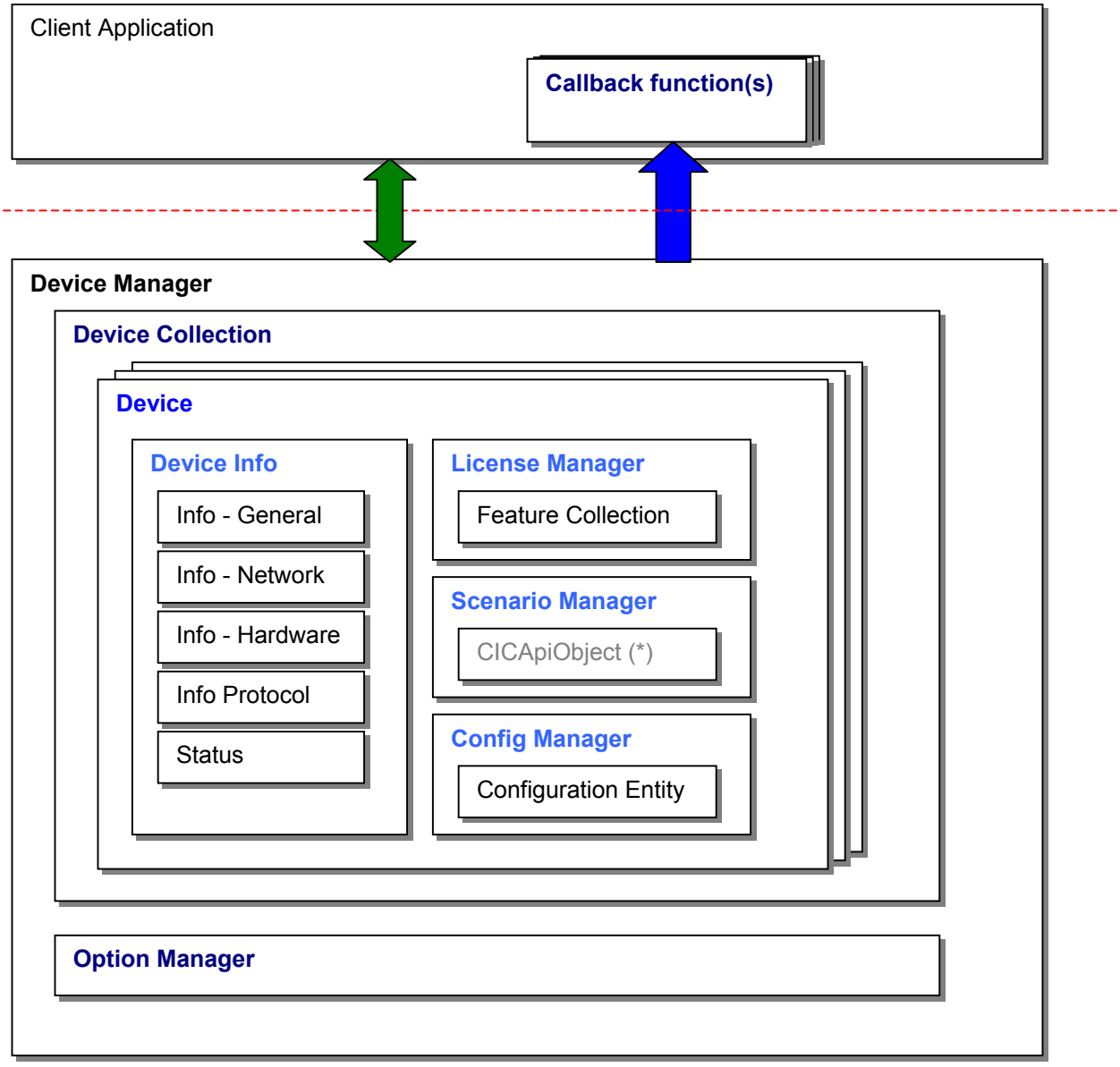
3.1. General

The Device Manager object main role is to control all aspects of the devices that are connected to the API instance. This includes:

- Listing all the devices that are discovered on the network.
- Getting and managing information about each device.
- Updating devices with new firmware and BusEngine files
- Updating devices with new configurations (impairment and monitoring scenarios)
- Updating license keys in devices and managing the features available for each device
- Handling notification about device status.

3.2. Programming Model

The device manager class is composed of several classes each responsible for a specific functionality. Figure 3-1 shows the hierarchy of the main classes used in the device manager. Additional classes and data structures are also incorporated in the device manager.



* The CICApiObject is defined as part of the Configuration Manager

Figure 3-1 Programming model

3.2.1. Definitions Data structures

The following is a list of the major definitions and data structures used in the Device Manger:

Name	Description
`INFNetMAC`	Represents the MAC address of a device. Another type, the INFDeviceID, is similar to the INFNetMAC definition.
`INFNetAddrIP`	Represents the TCP/IP address (v4 or V6) of a device
`INFDeviceTemperatureInfo`	Holds information about temperature of a device
`EINFDeviceState`	Definitions for the device connection status.
`EINFDeviceSessionState`	Definitions for the session status
`INFDeviceFeatureInfo`	Holds information about a specific feature that might be supported by the InFusion™ system.
`INFScenarioID`	Represents the ID of a scenario.
`EINFDeviceOptions`	Definitions for the application options

3.3. Device Manager

The devices manager main member is the CIBApiDeviceCollection object that manages a collection of InFusion™ device objects.

3.3.1. Functions List

Name	Description
`GetDeviceCollection`	Returns a pointer to the Device Collection object
`GetOptionManager`	Returns a pointer to the Options manager object

3.4. Device Collection

The Device Collection object, the CIBApiDeviceCollection, manages the collection of InFusion™ devices that were discovered on the network. The CIBApiDevice class represents the basic device entity.

A device can be accessed only through a pointer that can be obtained from the Device Collection.

On initialization of the SDK, The Device Collection attempts to load device information from a local device archive file named **INFDevices.archive**. The archive includes information about the devices name, user's notes and ID of device that were previously discovered. When the SDK's operation is terminated the Device Collection would store the current collection of devices into the archive.

Also, at given time intervals (default is every 5 minutes) the system would initiate a discovery process where the system probes for present InFusion™ devices on the network.

3.4.1. Functions List

Name	Description
`Refresh`	Refreshes the collections by starting a new discovery operation.
`GetDeviceCount`	Returns the total number of devices in the collection. This also includes devices that are not present on the network, but were read from the archive.
`ResetIterator`	Resets the iterator that allows to iterate through the collection of devices
`GetNextDevice`	Returns a pointer to the next device in the collection when iterating through he devices.
`GetPrevDevice`	Returns a pointer to the previous device in the collection when iterating through he devices.
`AddDevice`	Programmatically adds a device to the collection. This function should not be used in regular operation as the devices are added to the collection by the discovery mechanism.
`RemoveDevice`	Programmatically removes a device from the collection.
`GetDevice`	Returns a pointer to a device object.

3.5. Device

The device class, CIBApiDevice, corresponds to a single InFusion™ device. All functionalities for a device are available through this class. This includes:

- Control for a device's attributes (general, network, hardware, etc.) through a CIBApiDeviceInfo object.
- Control over device's configuration through a CIBApiDeviceConfigManager object.
- Control over device's license through a CIBApiDeviceLicenseManager object.
- Control over device's scenario configuration through the CIBApiDeviceScenarioManager object.
- Connecting to an InFusion™ device and disconnecting form it.
- Starting an impairment session and terminating it
- Resetting the InFusion™ device

3.5.1. Functions List

Name	Description
`GetDeviceInfo`	Returns a pointer to the device information object.
`GetDeviceConfigManager`	Return a pointer to the device's configuration manager.
`GetLicenseManager`	Return a pointer to the device's license manager.
`GetScenarioManager`	Return a pointer to the device's scenario manager.
`Connect`	Connects to the InFusion™ device.
`Disconnect`	Disconnects an existing connection to the device.
`StartSession`	Starts an InFusion™ session for the device.
`StopSession`	Stops an existing InFusion™ session.
`Reset`	Resets the InFusion™ device.

3.6. Device Info

The Device Info class, CIBApiDeviceInfo, includes several member objects, each representing a different group of the device's attributes.

3.6.1. Functions List

Name	Description
`GetID`	Return the unique ID of a device. The
`GetGeneralInfo`	Returns a pointer to the CIBApiDeviceInfoGeneral object that represents the device's general information.
`GetNetworkInfo`	Returns a pointer to the CIBApiDeviceInfoNetwork object that represents the device's network information.
`GetHardwareInfo`	Returns a pointer to the CIBApiDeviceInfoHardware object that represents the device's hardware information.
`GetProtocolInfo`	Returns a pointer to the CIBApiDeviceInfoProtocol object that represents the device's protocol-specific information.
`GetDeviceStatus`	Returns a pointer to the CIBApiDeviceStatus object that represents the device's status.

3.7. Device Info - General

The General device info class, CIBApiDeviceInfoGeneral, handles all the general devices information. This includes: Name and user's notes, protocol supported by the device and discovery information.

3.7.1. Functions List

Name	Description
`GetName`	Return a string that holds the name of the device.
`SetName`	Sets the name of the device.
`GetNotes`	Return a string that holds the user's notes.
`SetNotes`	Sets the user's notes.
`GetProtocol`	Returns the protocol that is supported by the specific InFusion™ device.
`GetProtocolText`	Return a text representation of the protocol supported by the InFusion™ device.
`GetDiscoveryInfo`	Returns a pointer to a CIBApiDeviceDiscoveryInfo object.

3.8. Device Discovery Info

The device discovery information object, CIBApiDeviceDiscoveryInfo, holds information about the means in which the device was discovered and added to the device collection.

3.8.1. Functions List

Name	Description
`WasDiscovered`	Return TRUE if the device was ever discovered during a discovery process. Returns FALSE if the device was added manually.
`GetUpdateTime`	Returns the date and time of the last time the device was discovered or added to the collection.
`GetDiscoveryMethod`	Returns the method by which the device was discovered (manually added to the collection or automatically discovered during a discovery process).

3.9. Device Info - Network

The network device info class, CIBApiDeviceInfoNetwork, handles all the network attributes of a device. This includes: TCP/IP address, MAC Address, IP Address mode (static or dynamic), subnet mask and connected host name.

3.9.1. Functions List

Name	Description
`IsIDsimilarToMAC`	Returns TRUE if the ID for the device is actually the MAC address of the device. This would happen when a connection to a device is established and the MAC address of the device is retrieved. This function returns FALSE when the MAC address of the device was never retrieved from the device. In this case a random unique ID is generated and assigned to the device record in the device collection. The random unique ID would be replaced with the real MAC Address once a connection is established.
`GetIPAddressMode`	Returns EINFDEV_IPADDRMODE_STATIC or EINFDEV_IPADDRMODE_DYNAMIC for static IP Address mode or dynamic, respectively.
`SetIPAddressMode`	Sets the IP address mode to STATIC or DYNAMIC.
`GetIPAddress`	Returns the IP address of a device as discovered.
`SetIPAddress`	Sets the IP address of a device.
`GetSubnetMask`	Returns the subnet mask of the device.
`SetSubnetMask`	Sets the subnet mask of the device.
`GetConnectedHostName`	Returns a string that holds the name of the host-machine that is connected to a device. If no host is connected to the device the returned string is empty.

3.10. Device Info - Hardware

The hardware device info class, CIBApiDeviceInfoHardware, handles all the hardware-related attributes of a device. This includes attributes like temperature, hardware failures and firmware failure.

3.10.1. Functions List

Name	Description
`IsHardwareFailureDetected`	Return TRUE is the device reports an internal hardware failure.
`IsTemperatureFailureDetected`	Return TRUE is the device reports a temperature exceeding a threshold level.
`IsFirmwareRunning`	Return TRUE if the firmware in the device is running correctly.
`GetTemperature`	Returns the temperature readout from the device

3.11. Device Info - Protocol

The device protocol info class, CIBApiDeviceInfoProtocol, controls the attributes that relate to the specific protocol the device supports.

3.11.1. Functions List

Name	Description
`GetOptions`	Returns an CICApiObject object that includes all the protocol attributes.
`SetOptions`	Sets the protocol attributes according to the CICApiObject object that is passed as an argument.

3.12. Device Status

The device status class, CIBApiDeviceStatus, reports the status of the device in two separated groups: the connection status of a device and the session status of the device.
The connection status can be one of the following states:

- `INFDEVSTATUS_NOT_PRESET` – The InFusion™ system cannot discover the deice on the network.
- `INFDEVSTATUS_BUSY` – The device is on the network, but is connected to another host.
- `INFDEVSTATUS_READY` – The device is on the network and ready to be connected to.
- `INFDEVSTATUS_CONNECTING` – The InFusion™ system is in the process of connecting to the device.
- `INFDEVSTATUS_CONNECTED` – The InFusion™ system is connected to the device
- `INFDEVSTATUS_DISCONNECTING` – The InFusion™ system is in the process of disconnecting from the device.

The session status can be one of the following:

- `INFDEVSESSIONSTATUS_UNKNOWN` – The session status of the device is unknown.
- `INFDEVSESSIONSTATUS_NONE` – There is no session running on the device
- `INFDEVSESSIONSTATUS_STARTING` – The device is in the process of starting a session.
- `INFDEVSESSIONSTATUS_IN` – The device is currently in a session.
- `INFDEVSESSIONSTATUS_STOPPING` – The device is in the process of stopping an existing session.

3.12.1. Functions List

Name	Description
`Reset`	Resets the status of a device to `INFDEVSTATUS_NOT_PRESET`
`GetDeviceState`	Return the connection status of a device
`IsPresent`	Returns TRUE if the device is present on the network.
`IsBusy`	Returns TRUE if the device is busy.
`IsReady`	Returns TRUE if the device is ready to be connected to.
`IsConnecting`	Returns TRUE if the InFusion™ system is in the process of connecting to the device.
`IsConnected`	Returns TRUE if the device is connected to the host.
`IsDisconnecting`	Returns TRUE if the InFusion™ system is in the process of disconnecting from the device.
`GetSessionState`	Returns the session state of the device
`IsSessionStarting`	Returns TRUE if the device is in the process of starting a session.
`IsInSession`	Returns TRUE if the device is in a session.
`IsSessionStopping`	Returns TRUE if the device is in the process of stopping a session.

3.13. Device Config Manager

The Device configuration manager class, CIBApiDeviceConfigManager, is responsible for controlling the configuration of a device. The configuration Manager can retrieve information about the firmware, boot code and BusEngine that are loaded into the device. It can also update new firmware and BusEngine versions to the device. To update the boot code the device has to be returned for service to LeCroy.

The configuration manager encapsulates a collection of three entities: the firmware, the BusEngine and the boot code.

3.13.1. Functions List

Name	Description
`GetEntity`	Returns a pointer to the entity which ID was specified
`UpdateEntity`	Starts the process of updating the entity in the device with a new one.

3.14. Device Config Entity

The device configuration entity class, CIBApiDeviceConfigEntity, represents a single entity in the device. The entity can be the firmware, the BusEngine or the boot code.

3.14.1. Functions List

Name	Description
`GetName`	Returns the name of the device, as was set by the user.
`GetDescription`	Returns the description of the device
`IsVersionRequiredARange`	Returns TRUE if the InFusion™ system supports a range of versions for this entity.
`GetVersionRequired`	Returns the required version for this entity. If the InFusion™ system supports a range of versions this value would be the first version of the range.
`GetVersionRequiredLast`	If the InFusion™ system supports a range of versions this function returns the last version of the range. Otherwise, the function returns 0x00000000.
`GetVersionCurrent`	Returns the version of the entity that is currently present in the device.
`GetFileName`	Returns the file name of the entity to be updated to the device.
`SetFileName`	Sets the file name of the entity to be updated to the device.
`GetFileExtension`	Returns the file extension of the entity to be updated to the device.
`GetVersionText`	Returns a string that holds a textual representation of the version.

3.15. Device License Manager

The license manager class, the CIBApiDeviceLicenseManager, controls all the operations and information related to the license and features of the InFusion™ product. The InFusion™ system provides a comprehensive list of features that depend on the license key downloaded to the box.

Only LeCroy generates license key files. Each file is suitable for a specific device.

3.15.1. Functions List

Name	Description
`GetFeatureCollection`	Returns a pointer to the features collection object.
`UpdateLicense`	Allows to update a license key to the device.
`IsMaintenanceValid`	Returns TRUE if the maintenance is still valid.
`GetMaintenanceEndDate`	Returns the last day of the maintenance period.

3.16. Device Feature Collection

The feature collection class, the CIBApiDeviceFeatureCollection, holds all the information related to the features that are enabled by the license key stored in the device.

3.16.1. Functions List

Name	Description
`GetFeatureCount`	Returns the total number of available features in the collection.
`ResetIterator`	Resets the iterator that allows to iterate through the collection of devices.
`GetNextFeature`	Returns pointer to the next feature data structure (INFDeviceFeatureInfo) when iterating trough the collection.
`GetPrevFeature`	Returns pointer to the previous feature data structure (INFDeviceFeatureInfo) when iterating trough the collection.
`GetFeature`	Returns pointer to a feature data structure (INFDeviceFeatureInfo) that has the given ID.

3.17. Device Scenario Manager

The scenario manager class, the CIBApiDeviceScenarioManager, includes all the functionality required to control he scenarios that are loaded to the device and retrieved from the box.

Up to 10 scenarios can be loaded to the box at a time.

Each Scenario has a unique ID that is assigned by the InFusion™ system when a scenario is created.

3.17.1. Functions List

Name	Description
`GetSelectedScenarioIndex`	Returns the 0-base index of a scenario that is going to be activated if a 'Start Session' command is issued to the device.
`SetSelectedScenarioIndex`	Sets the index of the scenario that is going to be activated if a 'Start Session' command is issued to the device.
`GetSelectedScenarioID`	Returns the unique ID of the selected scenario.
`GetDatabase`	Returns a pointer to a database object that includes all the scenarios that were retrieved from the device.
`SetDatabase`	Updates the device with a group of scenarios that are included in the database.
`GetScenario`	Retrieves a scenario object from a device and returns a corresponding scenario object.
`SetScenario`	Updates the device with a specific scenario.
`RemoveScenario`	Removes a scenario from a given slot. The slot becomes an 'Empty Slot'.

4. Monitor

4.1. General

The Monitor object main role is to provide streaming & logging capabilities for monitoring and system events. This includes:

- Events that originate from the BusEngine when running a scenario.
- System events about status of InFusion™ devices and general InFusion™ system notifications.

4.2. Programming Model

The monitor class main role is to provide control over the output streams.

An output stream may include logging information from one or more devices. The type of the messages can also be set for any stream, so the application gets full control over the characteristics of each stream.

Throughout the life cycle of an output stream, several InFusion™ sessions may start and stop. The client-application has full control over which session to get.

The monitoring stream is based on a simple callback function scheme, where the client application registers a callback functions when it calls to create a stream.

Figure 4-1 shows the hierarchy of the main classes used in the monitor. Additional classes and data structures are also incorporated.

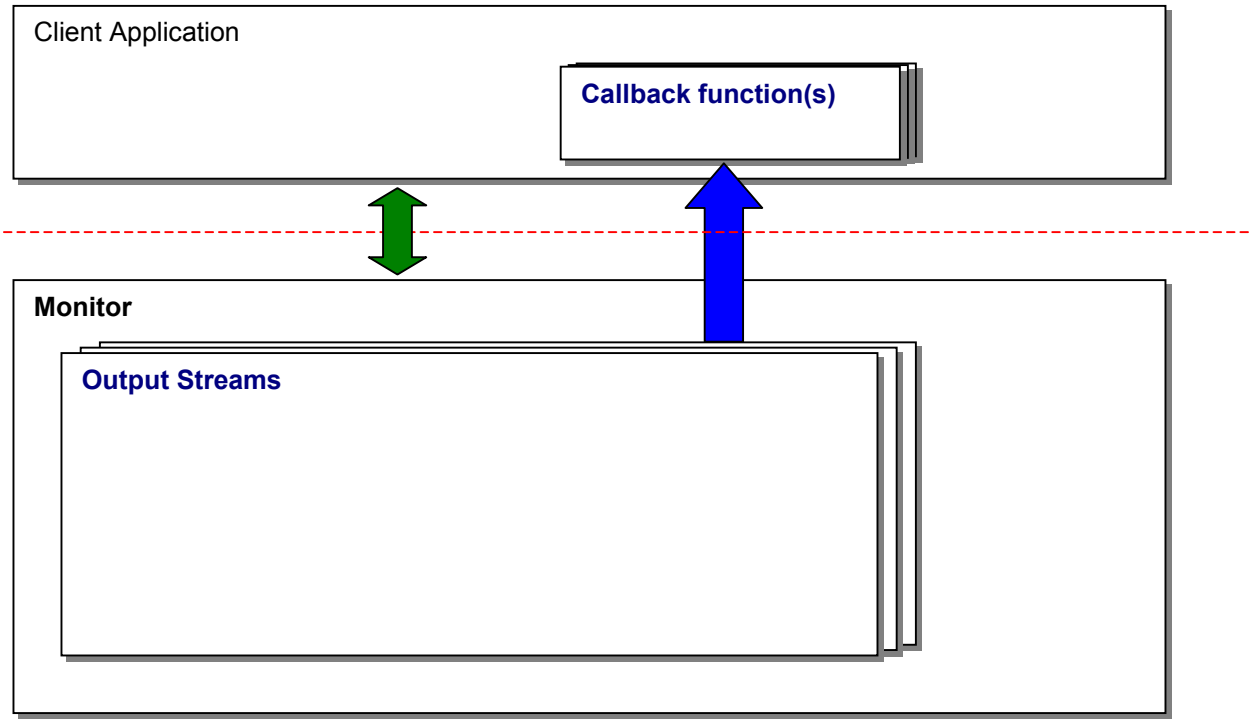


Figure 4-1 Monitor Programming model

4.2.1. Definitions Data structures

Name	Description
`MONITORING_FLAGS_*`	Flags for creating a stream.
`SIMApiTimeStamp`	Holds time stamp value.
`SIMApiMessage`	Holds information about a monitoring message
`EIMApiInfoType`	Type of information. Possible values are: EINFMON_INFOTYPE_ACTION, EINFMON_INFOTYPE_SNAPSHOT or EINFMON_INFOTYPE_OUTPUT.
`EIMApiMessageType`	Type of message. Possible values are: EINFMON_MSGTYPE_ERROR, EINFMON_MSGTYPE_WARNING or EINFMON_MSGTYPE_INFO
`SIMApiScenarioSource`	Holds information about the Scenario.
`SIMApiCounter`	Holds information about the counter data that is delivered.
`SIMApiCountersMapping`	Holds information about the counters mapping for the specific session running.
`SIMApiMessageHeader`	Header for the monitoring message
`SIMApiOutputMessage`	Holds information about an output message.
`SIMApiActionMessage`	Holds information about an action message.

4.3. Monitor

The monitor class, CIMApiMonitor, is the top-level class in the Monitor. It is responsible for controlling the output streams.

4.3.1. Functions List

Name	Description
`CreateMessagesStream`	Creates a monitoring stream to be directed to a callback function.
`PauseMessagesStream`	Pauses the streaming of messages to the client application's callback function.
`ResumeMessagesStream`	Resumes the streaming of messages to the client application's callback function.
`IsRunning`	Returns TRUE if the specified stream is enabled (and not paused).
`CloseMessagesStream`	Closes the specified monitoring stream and un-registers the client-application callback function.

4.4. Output Stream

The output stream class, IOutputStream, represents a single monitoring stream.

4.4.1. Functions List

Name	Description
`ProcessMessages`	This callback function is overridden by a client-application function.
`GetSelectedSession`	Returns the ID of the last session for the output stream.

5. Configuration Manager

5.1. General

The Configuration Manager object controls all the aspects of InFusion™ sessions. This includes:

- Maintaining libraries of scenarios, events, actions and states.
- Providing the translation of scenarios and their components into the Bus Engine “language”.

5.2. Programming Model

The Configuration Manager class' main role is to provide capabilities of working with Scenarios, creating, editing, saving, reading Libraries, translating Scenarios to the Bus Engine structures and preparing data to be sent to the InFusion Device.

InFusion Database (or InFusion Library) is a self-contained object. Client application can instantiate as many Library Objects as needed using capabilities provided by Configuration Manager. After the Library Object has been created client application becomes responsible for deletion of that object and can call all its methods to make use of it.

Figure 4-1 shows the hierarchy of the main classes used in the Configuration Component. Additional classes and data structures are also incorporated.

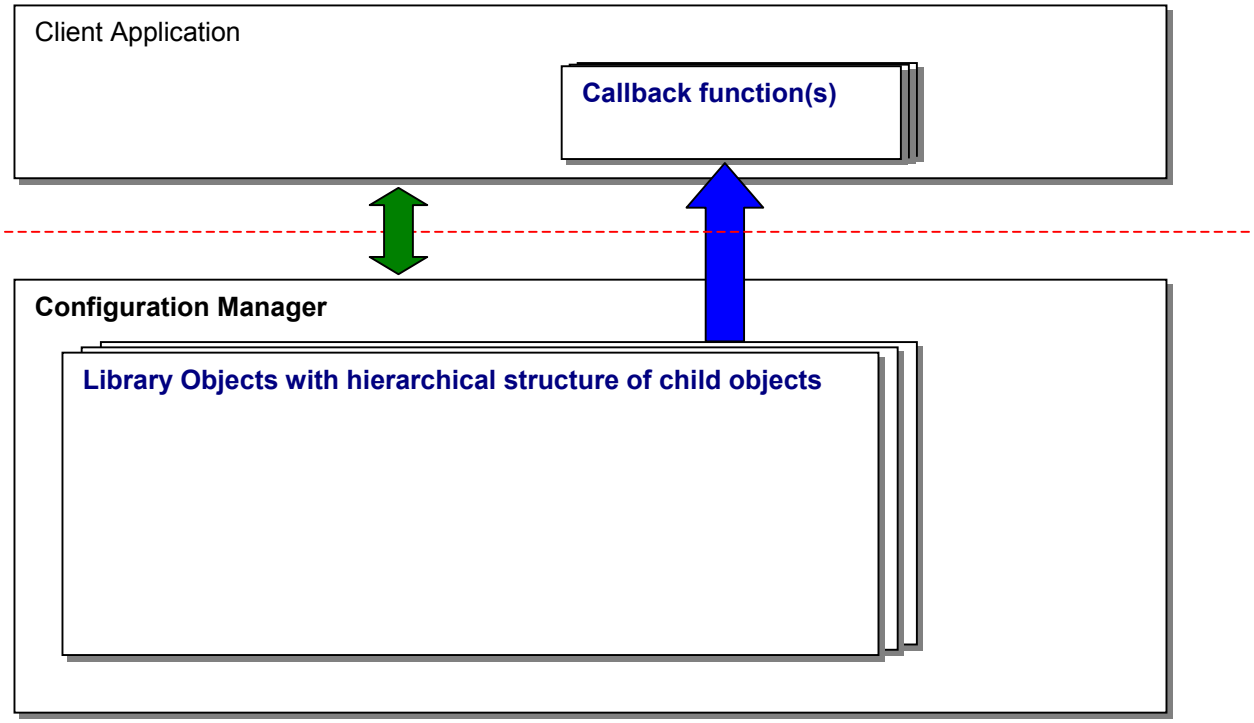


Figure 5-1 Configuration Manager Programming Model

5.3. Config Manager

The Configuration Manager class, CICApiConfigManager, is the top-level class in the Configuration Manager.

5.3.1. Functions List

Name	Description
`GetDatabase`	Returns a configuration database.
`CreateDatabase`	Creates and returna a new configuration database.
`PushServiceLevel`	**(For future use)** Pushes the service level to the service level stack.
`PopServiceLevel`	**(For future use)** Pops the service level to the service level stack.
`GetServiceLevel`	**(For future use)** Returns the current service level.
`GetICErrorText`	Returns a string holding a textual representation of the return code (see also CICApiErrorInfo in 5.6.)
`SetAppVersion`	Sets the version of the client application.
`SetAppBuild`	Sets the build number of the client application.

5.4. Object

The Object class, the CICApiObject, is the base class for all the configuration objects. Objects are organized hierarchically, forming complex graph with links and interrelations. There are two levels of differentiation of objects – ObjectType and ObjectSubtype. Each individual object has both. Certain relationships between objects are allowed or disallowed based on their types (for example Scenario Object can be a child object for Database Object, but never the other way around). Figure 5-2 shows an example of the relationships between objects.

Below is the list of currently supported Object Types:

```
enum EICObjectType
{
    IC_OBJTYPE_NONE = -1,

    IC_OBJTYPE_ACTION = 0,
    IC_OBJTYPE_COUNTER,
    IC_OBJTYPE_TIMER,
    IC_OBJTYPE_LOGIC,
    IC_OBJTYPE_BASIC_ELEMENT,
    IC_OBJTYPE_EVENT,
    IC_OBJTYPE_STATE,
    IC_OBJTYPE_GLOBAL_STATE,
    IC_OBJTYPE_SEQUENCE,
    IC_OBJTYPE_SCENARIO,
    IC_OBJTYPE_REPOSITORY,

    IC_OBJTYPE_DATABASE,

    IC_OBJTYPE_DEVICE_SETTINGS,

    IC_OBJTYPE_LAST
};
```

These types represent different nature of objects and allow different parts of interface functions of CICApiObject class to be meaningful.

Each important Object Type has its own list of subtypes that give more detailed differentiation of objects. Thus there are Event Types, Action Types, Logic Element Types and so on.

Object properties can be accessed/edited through Object Attributes. All attributes have string-based interface and can be set from text string and provide its text representation. See CICApiAttribute (chapter 5.5.) for more details.

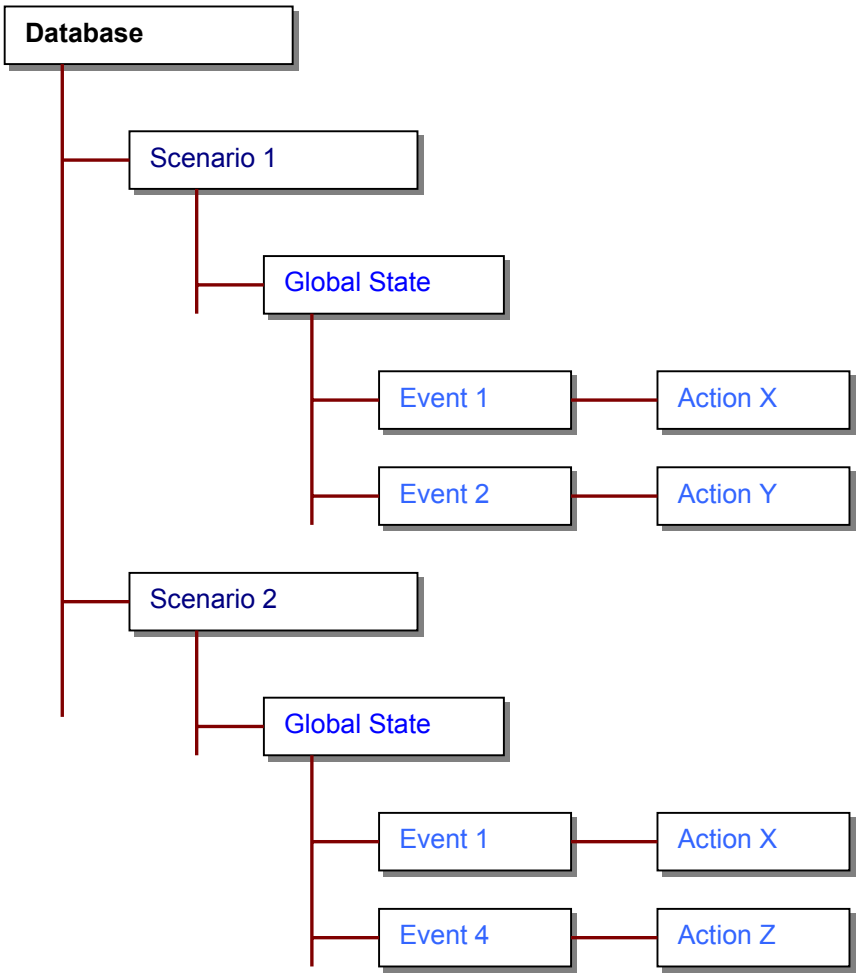


Figure 5-2 Relations between configuration Object

5.4.1. Functions List

5.4.1.1. General

Name	Description
`Initialize`	Initializes the object
`IsValid`	Checks object's validity and integrity
`IsTypeValid`	Verifies validity of Object Type
`HasFailedToLoad`	Indicates whether the object failed to load from database file or from the InFusion box. (applicable to Scenario Objects)
`GetDatabase`	Accesses default database
`CopyObject`	Copies content of the other object into this one. Only applicable to objects of the same type and subtype.
`CloneEx`	Creates a clone of the object
`CloneInPlace`	Creates a clone (brother) of the object under the same parent
`CloneDetached`	Creates a clone of the object without any parent
`GetText`	Gets text representation of the object
`IsEmpty`	Checks whether the object is empty (applicable to Scenario, State or Global State objects)
`IsEmptySlot`	Checks whether Scenario Slot (Scenario object in the Database representing some actual InFusion box) is empty
`GetDirectionAttr`	Returns pointer to CICApiAttribute responsible for direction
`GetDirection`	Gets Direction for traffic changes (for Scenario Objects) or for detection of events (for Event Objects)
`SetDirection`	Sets direction for traffic changes (for Scenario Objects) or for detection of events (for Event Objects)

5.4.1.2. Object Types and IDs

Name	Description
`GetObjectType`	Gets Object Type
`GetObjectSubtype`	Gets Object Subtype
`GetObjectId`	Gets Object Id (GUID)
`SetObjectId`	Sets Object Id (GUID)
`GetObjectIdText`	Gets Text Representation of Object Id
`GetBaseObjectId`	(For future use)
`SetBaseObjectId`	(For future use)
`GetNameAttr`	Returns pointer to CICApiAttribute holding object name
`GetName`	Gets Object Name

`SetName`	Sets Object Name
`GetDefaultName`	Gets Object Default Name
`GetDescriptionAttr`	Returns pointer to CICApiAttribute holding object description
`GetDescription`	Gets Object Description
`SetDescription`	Sets Object Description
`GetObjectTypeName`	Gets Object Type Name
`GetObjectTypeByName`	Gets Object Type by string Name

5.4.1.3. Attributes

Name	**Description**
`GetAllAttributes`	Gets list of object attributes
`GetAttribute`	Gets attribute by name
`AnyAttributeHasDependencies`	Checks attribute dependencies

5.4.1.4. Validation Errors List

Name	**Description**
`GetErrList`	Returns list of Errors for Scenario Validation. See also CICApiErrorInfo

5.4.1.5. XML Loading And Saving

Name	**Description**
`LoadFromFile`	Loads Database or Scenario object from file
`SaveToFile`	Saves Database or Scenario object to file
`SaveCopyToFile`	Saves Database or Scenario object with changed ID to file
`LoadFromString`	Loads Database or Scenario object from string
`SaveToString`	Saves Database or Scenario object to string
`GetCurVersion`	Gets Current Version of Object XML Saving Mechanism
`GetVersion`	Gets Version of Object XML Saving Mechanism
`GetCurProtocols`	Gets list of Currently Supported Protocols
`GetProtocols`	Gets list of Supported Protocols

5.4.1.6. Inputs/Outputs

Name	**Description**
`GetInputs`	Gets Object Inputs
`GetOutputs`	Gets Object Outputs

`IsOutputAllowed`	Checks if object of requested type is allowed to be an output of *this* object
`BindInput`	Binds another object as output (it becomes an output of *this*, and *this* becomes an input of the other object)
`BindOutput`	Binds another object as input (it becomes an input of *this*, and *this* becomes an output of the other object)
`UnbindInput`	Unbinds input
`UnbindOutput`	Unbinds output
`UnbindAllLinks`	Unbinds all inputs and outputs

5.4.1.7. Children Objects

Name	**Description**
`GetChildren`	Gets list of children
`IsChildAllowed`	Checks if object of requested type is allowed to be a child of *this* object
`CreateChild`	Creates object of requested type and subtype and adopts it as a child to *this*
`FindChildPos`	Find position of child object
`FindChild`	Finds child object
`DeleteChild`	Deletes child object
`ReplaceChild`	Replaces child object with another object (can be different type)
`Delete`	Deletes *this* object
`DeleteObject`	Deletes Object
`AttachChild`	Attaches child object
`DetachChild`	Detaches child object
`AdoptChild`	Adopts an object
`GetChildrenCount`	Gets Total Count of children of certain type
`GetIndex`	Gets zero-based index of the object among its siblings

5.4.1.8. Scenario Validation

Name	**Description**
`ValidateScenario`	Checks Validity of the Scenario
`GetDiagnostics`	Gets diagnostics in case ValidateScenario failed

5.4.1.9. Filtering Lists

Name	**Description**

`FilterChildren`	Filters children with specified criteria
`FilterInputs`	Filters inputs with specified criteria
`FilterOutputs`	Filters outputs with specified criteria

5.4.1.10. Parent Object

Name	**Description**
`GetParent`	Gets Parent Object
`GetRepository`	(For future use)
`GetGlobalState`	Gets Global State of the Scenario

5.5. Attribute

The attribute class, the CICApiAttribute, consists information for a single attribute of an object.

5.5.1. Functions List

Name	Description
`GetName`	Returns the attribute's name.
`GetType`	Returns the attribute's type.
`GetText`	Returns the attribute's value text
`SetFromText`	Containing a message to display. This function throws exception when failed.
`IsDefault`	Returns TRUE if the current value is the default value.
`IsUndefined`	Returns TRUE if the attribute doesn't have a value or the value is invalid.
`IsReadOnly`	Returns TRUE if the attributed should be grayed out in the dialog.
`CanEnterValue`	Returns TRUE if user should be able to type in the value.
`GetExpectFormat`	Returns attribute's expected format.
`GetStringMask`	Returns mask for editing the attribute.
`GetPredefinedStrings`	Returns list of strings representing predefined values.

5.6. Error Info

The error info class, the CICApiErrorInfo, holds information about error occurred during the Configuration Manager operation, for example during Scenario Validation. List of ErrorInfos is emptied every time before Validation procedure and filled up along the validation. Usually the list will have just a few errors, since critical errors stop Validation completely. Client application should always look through the error list to figure out what went wrong during Validation.

5.6.1. Functions List

Name	Description
`GetErrorDescription`	Returns detailed description of the Error.
`GetErrorObject`	Returns the pointer to a failed object (for example in Validation of Scenario, the objects that crossed the limits of Bus Engine resources limitations will be returned).
`GetErrorCode`	Returns the actual error code that was returned.
`GetErrorEntityName`	Returns the name of an entity that yielded the failure (for example in case of an Undefined Attribute of an object it'll return the name of that Attribute).