Gallagher
Controller API
(FTCAPI)

Interface Specification

Version: 7.10
Date: February 2013

Table of Contents

Disclaimer

Copyright

Version 5.20 Highlights

1. External System Identity

From version 5.20 of the Gallagher Controller API, each External System defined in Command Centre may now be assigned an identifier string. This is used to identify the particular External System through the API. The API now supports registration and deregistration notifications for External Systems, and not just External System Items as in previous versions. All inbound and outbound interface methods that reference a particular External System Item (ESI) now include a parameter that specifies the ESI's External System. This allows middleware to differentiate between multiple ESIs that may share the same identifier but attach to different External Systems.

2. External System and External System Item Configuration

It is now possible to assign a block of configuration text to External Systems and External System Items from within the Command Centre client. The API provides the assigned configuration text to the middleware when it notifies the middleware about the registration of these items. This happens as part of the new **notifySystemRegistered** and **notifyItemRegistered** notification methods.

Configuration data – like communications addresses, ports, names etc. – can now be assigned within Command Centre. This new functionality makes it possible to write self-configuring middleware that requires no user interface or file-based configuration support.

3. External System Types

The Command Centre user interface now allows an External System to be assigned a type. The Gallagher Controller API reports the type to the middleware as part of **notifySystemRegistered**. Note, however, that this is a separately licensable feature designed for use with the FT Middleware Framework. Unless the framework is licensed, the type selection, custom configuration UI, and reporting will be disabled.

4. Linking External Systems

Given the ability to identify External Systems at the API, version 5.20 also introduces the ability to link the identity of one External System to another. Thus, two External Systems defined within Command Centre may be linked such that they appear as the same system at the API. The two systems will take on the same identifier, configuration text, and type. Only one call to **notifySystemRegistered** will happen for the combined system.

This feature allows integrators to work around the limitation in the number of External System Items that may be assigned to an External System. For internal reasons that limit remains at 100, but now it is possible to assign extra items to a new system and link the system identities, creating one effective system at the API.

5. Interface Changes and Compatibility

Due to the changes in the way External Systems and ESIs are described at the Gallagher Controller API, version 5.20 deprecates the existing interfaces and introduces new versions, named as follows:

- **IFTExternalEventInput2**
- **IFTMiddleware2**
- **IFTTrigger2**

The new interfaces are now all custom interfaces, not accessible via **IDispatch**. The first is implemented by the API component and the latter two optionally implemented by the client and connected via the standard COM connection point scheme.

The deprecated interfaces remain supported, however, so that the new API may be used with existing middleware. The following table lists the supported configurations when mixing pre-5.20 and post-5.20 versions of software:

Middleware		Gallagher Controller API		Gallagher Command Centre and Controller	
Pre-5.20	5.20 and later	Pre-5.20	5.20 and later	Pre-5.20	5.20 and later
◆		◆		◆	
◆			◆		◆
	◆		◆		◆

Please note the following important implications from the above table:

a) It is **possible** to run old middleware against new API and Command Centre components.

b) It it **not possible** to run the old API component with an upgraded Command Centre system. This means that when upgrading Command Centre and Controllers to v5.20 and later, it is necessary to upgrade the API component at the same time. This should not require rework of existing middleware.

c) It is **not possible** to run a new version of the API with an old version of the Command Centre system components.

d) The valid upgrade procedure is: **first upgrade Command Centre to v5.20 or later, then upgrade the Controller and Gallagher Controller API at the same time.**

6. .NET

The version 5.20 release includes a COM Interop Assembly enabling the use of the API from a managed .NET application. The COM Interop Assembly is contained in the file `CardaxFTCAPILib.dll.`

Version 6.01 Highlights

1. Logging Card Events with Long Card Numbers from Gallagher Controller API

A new inbound interface named **IFTExternalEventInput3** has been introduced. The interface is a *custom* interface, not accessible via **IDispatch**. It is implemented by the API component to allow external systems to submit card events to the Gallagher Command Centre event system using long card numbers. The support of long card numbers is a new feature introduced in version 6.01 of Command Centre.

2. Dispatching Display Messages via Gallagher Controller API

A new outgoing interface named **IFTMessages** has been introduced. The interface is a *custom* interface, not accessible via **IDispatch**, optionally implemented by the client and connected via the standard COM connection point scheme. By implementing the interface, the client can receive the messages that are similar to informational and feedback messages as appear on the IDT and other readers with text display, (e.g. "Please Enter" and the "Secure" prompts). Unlike IDTs and RATs, where the number of characters that display is limited, secondary display devices via the Gallagher Controller API can display complete messages. The messages can only be generated when the External System Item is configured as an access reader, such as being assigned as one of readers on the Door item in the Command Centre system.

3. Compatibility

Version 6.01 does not support mixing pre-5.20 and post 5.20 versions of software. Therefore all software must be running v6.01 with the interface changes and compatibility from 5.20 implemented.

Interface Implementation

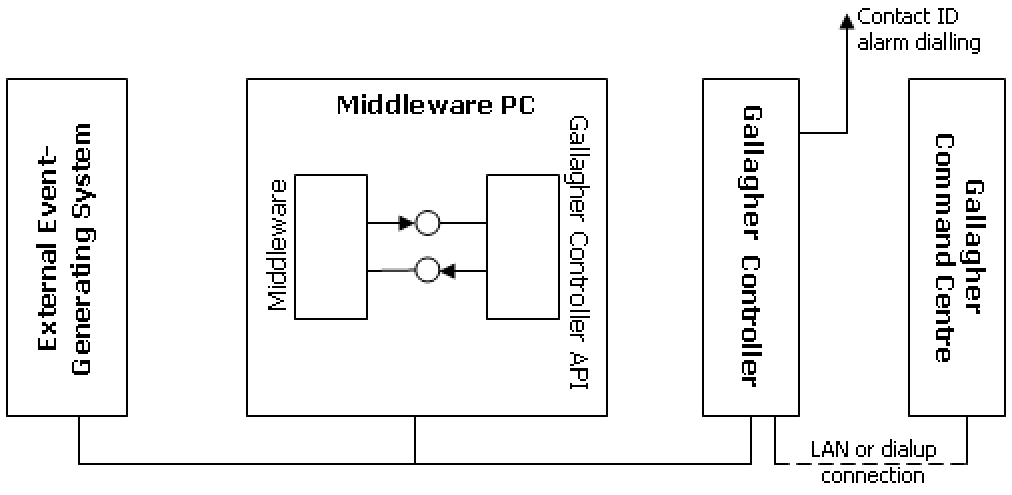
This section consists of the following topics:

Architecture

The Gallagher Controller API is designed as a standalone COM component providing simple interfaces by which an external system may submit event and status information to a Gallagher Command Centre system, and by which the Command Centre system may publish strings keyed to Command Centre event or alarm occurrence and output state changes.

The Controller API is designed to run on a middleware machine and look after all communications, protocol, licensing, and encryption details so as to make it very simple for middleware engineers to interface to the Command Centre system. If desired, the middleware machine may be the same machine as the Command Centre is running on. (See configuration notes later in this section.)

The Controller API allows external systems to easily integrate with Command Centre, including gaining a presence within the Command Centre workstation application, displaying text and graphical status to the Command Centre operators, and logging events and alarms into the Command Centre event system. It also allows external systems to take advantage of the Command Centre alarm management functionality, including alarm dialling and action plans.



Component

The Gallagher Controller API is implemented as a COM component with the following packaging:

Binary file (including type library):	**CardaxFTCAPI.dll**
COM Interop Assembly (for .NET):	**CardaxFTEIAPILib.dll**
COM Component Name:	**CardaxFTEIAPI**

The Gallagher Controller API component implements two inbound and three outgoing interfaces:

IFTExternalEventInput2	An inbound interface that allows event, event restoral, and status information to be submitted to the target Command Centre system.
IFTExternalEventInput3	An inbound interface that allows card event and event restoral information to be submitted to the Command Centre system. This is similar to the above interface except that the events contain long card numbers.
IFTMiddleware2	An outgoing interface that allows client middleware to receive notification of External System and External System Item registration at the API, and notification of alarm acknowledgement by Command Centre operators. Client implementation of this method is optional. Gallagher Controller API connection to the client's implementation of this method is via the standard COM connectable objects strategy.
IFTTrigger2	An outgoing interface that allows client middleware to receive notifications and configurable strings triggered by event/alarm occurrence or output state changes in Command Centre. Client implementation of this method is optional. Gallagher Controller API connection to the client's implementation of this method is via the standard COM connectable objects strategy.
IFTMessages	An outgoing interface that allows client middleware to receive access prompts that are similar to those appearing on IDTs or other readers with text display. Client implementation of this method is optional. Gallagher Controller API connection to the client's implementation of this method is via the standard COM connectable objects strategy.

Refer to the interface reference documentation later in this specification for detailed information about these two interfaces and their use, including the blocking nature of calls on the **IFTExternalEventInput2** and **IFTExternalEventInput3** interface.

The Controller API component also implements the following interfaces:

IUnknown	Mandatory base interface.
IConnectionPointContainer **IConnectionPoint** **IEnumConnectionPoints** **IEnumConnections**	These standard COM interfaces are implemented by the Gallagher Controller API or by connection or enumerator sub-objects according to the COM connectable object strategy. They all support the ability of the client middleware to connect its **IFTMiddleware2**, **IFTTrigger2** and **IFTMesssages** interface implementations to the Gallagher Controller API component.

Licensing

The Gallagher Controller API is a licensable feature for Gallagher Command Centre installations. Every Command Centre installation requires a Gallagher license file, which is keyed to specific Controller hardware running on the site.

Licensing details are largely transparent to clients of the Controller API. **If the API is installed on a separate machine from Command Centre server, the same license file that is installed on the server (in the C:\Windows directory) must be present on the API machine.** The license files must match each other, otherwise the API will not communicate with the target system. Given that these conditions are met, and given that the Gallagher Controller API feature appears in the license file, then the client code does not need to be concerned with further licensing details.

If, however, there is a licensing problem – an expired license, a missing license, a non-matching license, a corrupt or tampered license – then calls to methods on the I**FTExternalEventInput2** or **IFTExternalEventInput3** interface will fail with one of two return codes:

Return Code	Situations	Explanation
FTCAPI_E_LICENSE	Tampered, corrupt, expired or absent license file. Gallagher Controller API feature not listed in license file.	The license file *failed* the various validation checks performed by the Gallagher Controller API component.
E_PENDING	Legitimate, intact license file containing the Gallagher Controller API feature, but does not match the license in use by the target Command Centre system.	The license file *passed* the validation checks performed by the Gallagher Controller API component. But because the license does not match the one in use by the target system, a communication failure results. The Gallagher Controller API sees itself as running standalone waiting for the target system to contact it. It therefore has no event source registrations on it, and hence the E_PENDING return.

Security

Gallagher Controller API security can be assessed on three levels:

1. *Configuration Security*

   A Gallagher Controller API instance may not be used to submit events and status information into a Gallagher Command Centre installation unless the relevant External System and External System items have first been created and configured within the Command Centre system. This includes assigning the External System a specific middleware PC. Similarly, events and status information *may not* be submitted against configured sources from an Gallagher Controller API instance running on any machine other than the one configured against the sources' External System.

   Event and status submission through the Gallagher Controller API into a Command Centre system is thus secured by the same operator-level security used for the rest of the Command Centre system configuration, including operator assignments, operator groups, operator privileges. Attention should therefore be paid to operator privilege assignments, in particular the "Edit Site" privilege.

2. *Communications Security*

   Communications between the middleware/Gallagher Controller API PC and the rest of the Command Centre system are secured through standard Command Centre encryption and authentication policies. The level of encryption is determined by the purchased level for the individual site, which is specified in the site's Command Centre license file.

3. *Code Security*

   The Gallagher Controller API is a COM component implementing one ingoing and one outgoing interface. There are no authentication methods specifically built into the interfaces. *It is the responsibility of the installing site to secure the PC that has been designated to run the middleware and Gallagher Controller API instances.* This includes standard procedures such as virus protection and administering the COM/DCOM access permissions for the Gallagher Controller API component. The security of the designated middleware/Gallagher Controller API machine is vital in preventing unauthorised software launching the Gallagher Controller API and submitting event and status information to the system.

Implementation Notes

1. *Error Codes and Visual Basic*

   Each of the API interface methods return an HRESULT to indicate method result. When Visual Basic applications make COM calls, Visual Basic automatically intercepts the HRESULT return and uses it to throw an exception. The HRESULT return is then available to the Visual Basic application in the intrinsic global **Err** object, **Number** property. In Visual Basic .NET you must catch a **`System.Runtime.InteropServices.COMException`** and the **`HRESULT`** will be in the **`ErrorCode`** property.

2. *IDispatch*

   For backward compatibility, the deprecated interface **IFTExternalEventInput** remains as the default interface on the Gallagher Controller API component (named CardaxFTEIAPI.) This interface is a dual interface and so supports calling via **IDispatch**. Likewise, the two deprecated outgoing interfaces are dispinterfaces, with **IFTMiddleware** marked as the default source interface. Scripting clients and clients written with Visual Basic 6 or earlier should use these interfaces.

   The interfaces – **IFTExternalEventInput2, IFTMiddleware2, IFTTrigger2** (introduced with version 5.20), **IFTExternalEventInput3** and **IFTMessages** (introduced with version 6.01) – are all custom interfaces inheriting from **IUnknown** only. If the middleware is to be written with Visual Basic, use a recent version that is capable of calling and implementing custom interfaces.

3. *.NET Support*

   Middleware may be written using the .NET Framework. In this case the middleware uses the API via the included COM Interop assembly, which defines .NET versions of the API interfaces. .NET client code may connect to the API's outgoing interfaces either by casting the API object to **IConnectionPointContainer** and calling **Advise** on the obtained connection point, or by using .NET events via the delegates defined in the Interop assembly.

API Location

Beginning with version vEL5.10 of Gallagher Command Centre, it is now possible to locate the Gallagher Controller API on the same machine as the Command Centre server components. This scenario is not enabled by default, however, because the Gallagher Controller Server and the Gallagher Controller API both attempt to use the same IP port for communications.

To enable the co-location of the Gallagher Controller API with the Gallagher Controller Server it is necessary to first reconfigure the Gallagher Controller server to use a different IP port for its communications. Use the following procedure:

1. Stop the Command Centre services.
2. Open up the registry editor and browse to **`HKEY_LOCAL_MACHINE\SOFTWARE\Gallagher\Command Centre`**
   Note: If using a 64 bit system, browse to **`HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Gallagher\Command Centre`**
3. Create a new DWORD value and rename it to **`ListenPort`**.
4. Edit the value of **`ListenPort`** by selecting the decimal radio button and then typing in the desired port number.
   Note: The valid range is 1024 to 65535. Do not try to use port numbers outside of that range.
5. Start the Command Centre services and logon to Gallagher Command Centre.
   All of the Controllers will appear as offline and will not come online by themselves.
6. Select 'Push Configuration' on each Controller in turn.
   They should each come online after they are pushed.

Notes:

- If multiple Gallagher Controller communications servers are being used then Steps 1 - 4 need to be repeated for each one.
- When using a port number other than the default of 1072, if a Controller is restarted with DIP switch 2 on, then it will be unable to communicate with the server until an operator selects 'Push Configuration' on that Controller. This causes a connection to be initiated by the server rather than from the Controller and tells the Controller what port number to connect to.
- The command **`"netstat -an"`** can be used to check that the Controller service is indeed now listening on a different port number, or after Step 1 to initially check that some other program is not already listening on the port number you intend to use.
- All Controllers and the Gallagher Controller API all still talk to each other using port 1072. It is only communication with the server that is now on a different port.

Interface Application

This section consists of the following topics:

External Systems and External System Items

External System

Gallagher Command Centre defines a generic item type to represent an external system that connects to it. This is the *External System*, created and configured through the External Systems master list window in Command Centre. An External System can be thought of as a logical item used to group together several external event-generating sources, all of which communicate through a single API instance.

Currently the following configuration is required on an External System:

1. *Gallagher Controller*	A single Gallagher Controller unit that is 'home base' for the external system. This is used internally for data transfer and alarm management functions and should be on the same local network as the External System's middleware PC and API.
2. *Middleware PC*	The name or address of a machine that runs the middleware software and Gallagher Controller API instance serving the External System.
3. *Identifier*	A text identifier, user-defined and unique across all External Systems, used to identify the External System at the Gallagher Controller API.

Command Centre uses the first two properties to determine where to find, and how to talk to, the Controller API instance responsible for serving the particular External System.

It is legitimate to configure several External Systems with the same middleware PC. This implies they will all be served by the same API instance.

It is also legitimate to configure the middleware PC to be the same machine used to run a Gallagher Controller Server process, **as long as the Controller Server has first been configured to use a different TCP/IP listening port from its default, normally 1072**. This must be performed first, and all Controllers refreshed to point to the new server port, before attempting to install and run the API on the same machine. (See ***API Location*** on page 15.)

In addition to the above, it is possible to specify the following on an External System:

1. *Configuration*	A block of text, free-format and containing a maximum of 1024 characters, that is given to the middleware at API start- up as part of **IFTMiddleware2::notifySystemRegistered**. This may be used to provide the middleware with configuration information for the system, such as addressing etc., obviating the need for custom configuration support in the middleware.

2. *Identity Same As*	It is possible to specify that an External System will take the same identity as another External System defined in Gallagher Command Centre. This means that the system will take the same *identifier*, *type*, and *configuration text* as the selected system. The External Systems that are linked like this will appear as a single system at the API, causing only one call to **notifySystemRegistered** for all systems that share the identity. This feature enables a work around for the limitation on the number of External System Items that may be assigned to one External System. For internal reasons that limit remains at 100.

External System Item

Each External System can have up to 100 External System Items assigned to it. An External System Item (ESI) represents a single event-generating entity against which is configured a name, description, and an event response program used to control alarms and their consequences.

Currently the following configuration is required on an External System Item to achieve event logging against it:

1. *External System*	A single External System item representing the system to which the source is attached. This determines where the source is registered, (i.e. which middleware/API instance will know about the source), and therefore determines where events may be logged against the source.
2. *Identification*	A text identifier, user-defined and unique across all ESIs that connect to one External System, used to identify the ESI at the Gallagher Controller API.

Events and status may only be logged against an External System Item if the above configuration is present and valid within the Command Centre system, and the ESI has been registered at the Gallagher Controller API, which is notified via **IFTMiddleware2::notifyItemRegistered**.

In addition to the above, it is possible to specify the following on an ESI:

1. *Configuration*	A block of text, free-format and containing a maximum of 7168 characters, that is given to the middleware at API start-up as part of **IFTMiddleware2::notifyItemRequired**. This may be used to provide the middleware with configuration information for the item, such as addressing, etc., obviating the need for custom configuration support in the middleware.

Alarm Handling

Events and Alarms

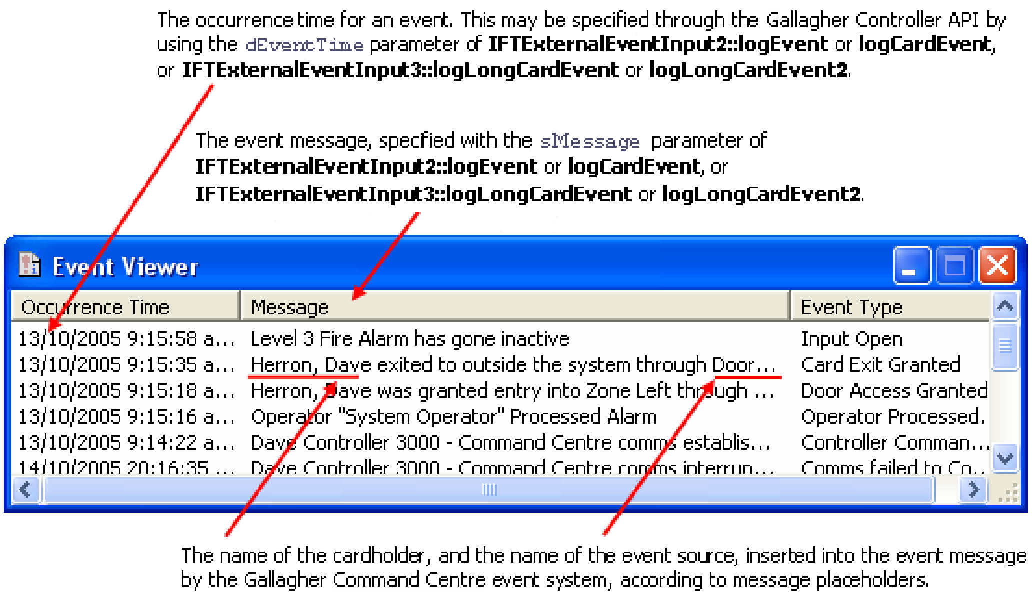
The Gallagher Command Centre system defines a set of events and event groups to cover a wide range of system scenarios. When an event is logged a range of tests are performed to determine whether the event should be elevated to an alarm. The tests include looking at the event group that the event belongs to, the configured Action Plan for the event group at the event source, and the state of the alarm zone for the event source, etc.

If an event becomes an alarm it gains a presence in the alarm window, along with many other alarm management features. Once in the alarm window an alarm requires *acknowledgement* by a Command Centre operator before it can be removed from sight.

The Gallagher Controller API allows client middleware applications to log *events* into the Gallagher Command Centre system. It does not, however, allow clients to programmatically specify the alarm priority of an event. This function remains the responsibility of Command Centre, working with its user-configured rules.

The Controller API currently allows any of 10 external event types to be logged through the interface, with identifiers from 0 to 9. Within Command Centre, each of these event types belong to a separate event group. So given that event response is configured at the event group level, full flexibility in alarm configuration is achievable.

When submitting these external event types through the Controller API it is possible for the caller to specify a text description to be displayed to the user wherever the event is displayed – for example in the event window or activity report. This text description may contain placeholders into which the system will insert such things as the name of the event source or the name of the event's cardholder.



The occurrence time for an event. This may be specified through the Gallagher Controller API by using the `dEventTime` parameter of **IFTExternalEventInput2::logEvent** or **logCardEvent**, or **IFTExternalEventInput3::logLongCardEvent** or **logLongCardEvent2**.

The event message, specfied with the `sMessage` parameter of **IFTExternalEventInput2::logEvent** or **logCardEvent**, or **IFTExternalEventInput3::logLongCardEvent** or **logLongCardEvent2**.

Event Viewer

Occurrence Time	Message	Event Type
13/10/2005 9:15:58 a...	Level 3 Fire Alarm has gone inactive	Input Open
13/10/2005 9:15:35 a...	Herron, Dave exited to outside the system through Door...	Card Exit Granted
13/10/2005 9:15:18 a...	Herron, Dave was granted entry into Zone Left through ...	Door Access Granted
13/10/2005 9:15:16 a...	Operator "System Operator" Processed Alarm	Operator Processed.
13/10/2005 9:14:22 a...	Dave Controller 3000 - Command Centre comms establis...	Controller Comman...
14/10/2005 20:16:35 ...	Dave Controller 3000 - Command Centre comms interrup...	Comms failed to Co...

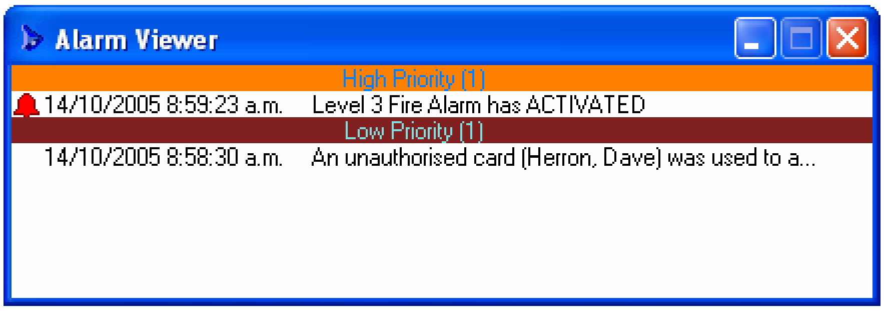
The name of the cardholder, and the name of the event source, inserted into the event message by the Gallagher Command Centre event system, according to message placeholders.

Note: It is not currently possible to log proprietary Gallagher Command Centre event types through the Gallagher Controller API interface.

Alarm State

The Command Centre event system supports both transitory events and 'stateful' events. *Transitory* events are momentary happenings recorded in the system as an event or alarm and able to be processed immediately. An example is a cardholder's unauthorised attempt to access a particular site area.

Stateful events retain either an active or an inactive state. Typically the alarm is generated when the state transitions from inactive to active. The stateful alarm may not be processed out of the Command Centre alarm window until the alarm condition goes inactive. Active stateful alarms show with an alarm bell icon beside them in the alarm window, as shown below:



Alarm Viewer

High Priority (1)

🔔 14/10/2005 8:59:23 a.m. Level 3 Fire Alarm has ACTIVATED

Low Priority (1)

14/10/2005 8:58:30 a.m. An unauthorised card (Herron, Dave) was used to a...

The Controller API allows both transitory and stateful events to be logged and updated:

Transitory Events	Use the **IFTExternalEventInput2::logEvent** method, **IFTExternalEventInput2::logCardEvent** method, **IFTExternalEventInput3::logLongCardEvent** method or **IFTExternalEventInput3::logLongCardEvent2** method to log, setting the **`bHasRestoral`** parameter to FALSE.

Stateful Events	Use the **IFTExternalEventInput2::logEvent** method, **IFTExternalEventInput2::logCardEvent** method, **IFTExternalEventInput3::logLongCardEvent** method or **IFTExternalEventInput3::logLongCardEvent2** method to log, setting the `bHasRestoral` parameter to TRUE. When the condition goes inactive (returns to normal) call **IFTExternalEventInput2::notifyRestore** to inform the system. This will remove the alarm bell 'active' indication and allow the alarm to be acknowledged and processed. Pay attention to the documentation for **notifyRestore**, in particular the need to pass the original event type, and the need to log the return-to-normal condition separately, if required.

Alarm Acknowledgement

It is possible for client middleware to receive notification when particular alarms are acknowledged by a Command Centre operator. Notification is received via the Gallagher Controller API's **IFTMiddleware2** outgoing interface and its **notifyAlarmAcknowledged** method. An example application may be the automatic reset of CCTV monitors when an operator acknowledges an alarm.

There are, however, several points to note about receiving alarm acknowledgements through the Gallagher Controller API:

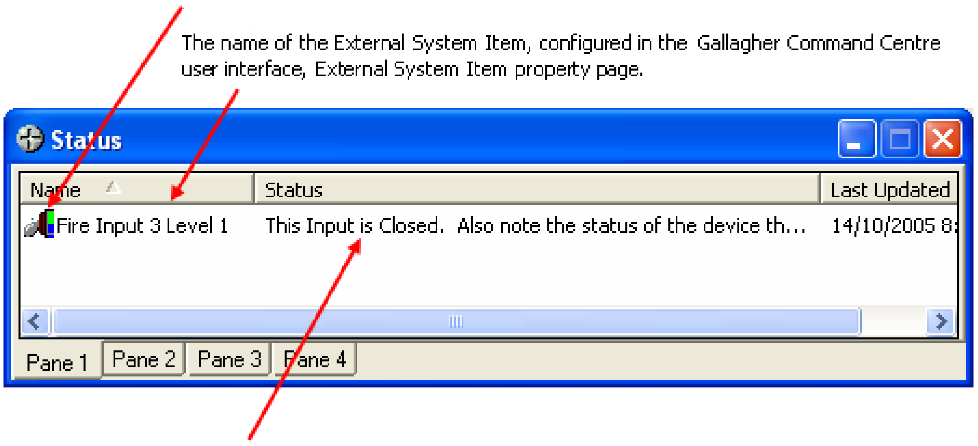
1. The alarm for which notification is required must have been originally logged through the Gallagher Controller API interface. This of course implies that the ESI is correctly configured and registered on the Gallagher Controller API with a functioning and licensed system.
2. The event logged through **logEvent**, **logCardEvent**, **logLongCardEvent** or **logLongCardEvent2** must be configured to generate an alarm within the Command Centre system. If the combination of event's group, its source, the action plan, and alarm zone has not been configured to generate an alarm for the logged event type, then *no alarm*, *alarm acknowledgement*, or *alarm acknowledged notification will follow*.
3. The event logged through **logEvent**, **logCardEvent**, **logLongCardEvent** or **logLongCardEvent2** must have its `lEventID` parameter set to something non-zero. It is the responsibility of the client middleware to create and track event IDs. Typically these will uniquely identify an event occurrence for which acknowledgement notification is required. If acknowledgement notification is not required for a logged event then set `lEventID` to 0.
4. Acknowledgement notification is not guaranteed by the Gallagher Controller API. There are some other internal situations which could cause client middleware to not receive alarm acknowledgement notification for a logged alarm. These are rare (for example Controller reset due to code upgrade) but never-the-less are possible. Client applications should take this into account.

Status

The Gallagher Controller API supports the setting and update of item status for each of the External System Items. Item status in Command Centre is displayed in icon form, wherever the item is visible, and in expanded text form in the Status Viewer utility, item's property page, and item's list window. Use the **IFTExternalEventInput2::notifyStatus** method to supply the system with status information, as shown here:

The External System Item icon, defaulting to a toggle switch but able to be customised through Gallagher Command Centre user-selectable icon sets. Icon state is controlled through the **IFTExternalEventInput2::notifyStatus** method, `lState, bTampered, bOffline` parameters.

The name of the External System Item, configured in the Gallagher Command Centre user interface, External System Item property page.



Name	Status	Last Updated
Fire Input 3 Level 1	This Input is Closed. Also note the status of the device th...	14/10/2005 8:

The status descriptor, supplied to Gallagher Command Centre through the **IFTExternalEventInput::notifyStatus** method `message` parameter

Triggered Strings

The Gallagher Controller API supports the dispatch of user-configured strings to the middleware via the **IFTTrigger2** outgoing interface. Strings may be related to either of the following activities:

Outputs	User-configured strings may be defined against every Command Centre Output in the system - one for the Output's closed/on state, and one for the Output's open/off state. Then, if the Output has been included in an External System Item's "Trigger - Output" list, the middleware will be notified whenever that output changes state. Specifically, the Gallagher Controller API will call the **IFTTrigger2::triggerOutput** method, passing the idenitification string for the External System Item and the Output's user-configured string that matches the state the Output is entering.
Events	One user-configured "event" string may be defined on every External System Item. Then, if the External System Item has been included in the "Cameras, Macros, and External System Items" list on an Action Plan, the middleware will be notified whenever that Action Plan fires. Specifically, the Gallagher Controller API will call the **IFTTrigger2::triggerEvent** method, passing the identification string for the External System Item and the External System Item's "event" user-configured string. This works for any priority level, (i.e. alarms **and** events). Also, the user-configured string may contain placeholder parameters (covering various event details and control characters) that will be substituted with their appropriate values at the time the **triggerEvent** method is called.

Reflect Outputs

The **IFTExternalEventInput2::notifyStatus** method may be used to set the state of Command Centre Outputs within the Command Centre system. This is achieved by using the *Reflect Outputs* feature. Each ESI has a *Reflect Outputs* property page as part of its properties. Gallagher Outputs may be dragged into the appropriate grid – one grid for straight reflection, one grid for inverted reflection. After applying this configuration, status setting via **notifyStatus** will cause the assigned Command Centre Outputs to switch state to reflect the state (or reflect the inverse of the state) of the specified External System Item.

Card Access from Gallagher Controller API

External System Items may be used anywhere in Command Centre in place of normal card readers. For example, on the properties of a Door item in Command Centre it is possible to assign entry readers and exit readers. By assigning an ESI here instead of a normal reader then the Gallagher Controller API may be used in place of a reader to provide card access information to the door. Card access decisions may therefore be made using the **IFTExternalEventInput2::logCardEvent** method, **IFTExternalEventInput3::logLongCardEvent** method, or **IFTExternalEventInput3::logLongCardEvent2** method.

Access Response Messages via Gallagher Controller API

The Gallagher Controller API supports the dispatch of access prompt messages to the middleware via the IFTMessages outgoing interface. When the External System Item is configured as an access reader, such as being assigned as one of the readers on the Door, the access prompts will be dispatched to the middleware upon card accesses. In this case, the Gallagher Controller API will call the **IFTMessages::displayMessage** method by passing the message identifier and/or relevant parameters, such as Cardholder name and/or Competency name, message and expiry due date (or expired date). The External System or ESI may be configured with appropriate messages to match each message identifier from the interface method.

IFTExternalEventInput2

The **IFTExternalEventInput2** interface is provided on the Gallagher Controller API component to allow external systems to submit event and status information to the Command Centre event and status systems.

Method	Description
`logEvent` (on page 28)	Log an external event into the Command Centre event system.
`logCardEvent` (on page 33)	Log an external card event into the Command Centre event system.
`notifyRestore` (on page 37)	Notify the Command Centre event system about a condition that has returned to normal. The restore should correspond to a condition previously logged with **logEvent** or **logCardEvent**.
`notifyStatus` (on page 40)	Notify the Command Centre status system about a change in status.

IFTExternalEventInput2::logEvent

Log an external event into the Gallagher Command Centre event system.

```
[VC++]
HRESULT logEvent( [in] LONG lEventType, [in] LONG lEventID,
                  [in] DATE dEventTime, [in] BOOL bHasRestoral,
                  [in] BSTR sSystemID, [in] BSTR sItemID,
                  [in] BSTR sMessage, [in] BSTR sDetails);


[VB]
Sub logEvent( lEventType As Long, lEventID As Long, _
              dEventTime As Date, _
              bHasRestoral As Boolean, _
              sSystemID As String, sItemID As String, _
              sMessage As String, sDetails As String  )
```

Parameters

`[in] LONG lEventType`

The event type code of the event occurrence. This should be within the range 0 to 9.

`[in] LONG lEventID`

A caller-defined number identifying the specific event occurrence. Assuming that the event is configured to generate an alarm within Command Centre, this identifier will be included as part of the **IFTMiddleware2::notifyAlarmAcknowledged** callback at such a time as a Command Centre operator acknowledges the alarm. Typically, this identifier will uniquely identify an event occurrence, although it is not required to. Set this parameter to 0 if you do not require notification of alarm acknowledgement.

`[in] DATE dEventTime`

The date and time of the event occurrence, in GMT. This parameter allows delayed events to be logged with their actual occurrence time. Fractions of a second will be ignored.

`[in] BOOL bHasRestoral`

Indicates whether the event is ‘stateful’ and therefore has a corresponding return-to-normal, or restoral, event.

`[in] BSTR sSystemID`

The string descriptor identifying the External System that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System Item in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sMessage`

The event message to be logged and displayed for the event within the Gallagher event system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sDetails`

The event details string to be logged and displayed for the event within the Gallagher event system. The string can be a maximum of 128 Unicode characters.

Return Values

This method supports the standard return values E_OUTOFMEMORY and E_UNEXPECTED, as well as the following:

S_OK

The event has been logged successfully to the Gallagher event system.

E_INVALIDARG

One or more arguments are invalid, including:

- **`lEventType`** out of the range 0 to 9.
- **`dEventTime`** out of acceptable range – refer to Remarks, below.

E_PENDING

The Gallagher Controller API is not ready to accept events against the specified source. The reasons may include:

- A corresponding External System Item, with a matching identifier, has not been configured in the Command Centre system.
- A corresponding External System has not been correctly configured in the Command Centre system.
- There is some communication problem between the Gallagher Controller API machine and the rest of the Command Centre system, possibly including its Controller hardware. Note that a communication problem could be due to the use of a license file that is unmatched to

the rest of the Command Centre system, even if that file is a correct and complete license in its own right.

- The Gallagher Controller API component has not had time to establish communication with the rest of the Command Centre system and receive registration details for the specified source.

FTCAPI_E_LICENSE

There is a problem with the Command Centre licensing. This may include:

- There was no Command Centre license file found in the Gallagher Controller API installation.
- The license file does not specify the Gallagher Controller API as a licensed feature for this site.
- The license file is tampered or corrupt.

FTCAPI_E_NETWORK

There was a network failure reaching the rest of the Command Centre system.

Remarks

logEvent is a blocking call used to log the details of an event occurrence into the Command Centre event system. The method will not return until either the event has been successfully logged to the system (which does not necessarily include reaching the Command Centre user interface) or else some failure condition is present. Failure conditions may also involve blocking, especially FTCAPI_E_NETWORK, which indicates a network timeout or otherwise unresponsive destination.

In order for **logEvent** to successfully process a submitted event, the specified source for the event must be registered with the Gallagher Controller API component. An event source is registered with Gallagher Controller API by having a matching External System Item correctly configured in Command Centre, and having the functioning, communicating, and licensed Command Centre system running.

There may be a delay between the starting of a Gallagher Controller API component and the point at which event sources are automatically registered on the component. Calls to **logEvent** for the event source will return E_PENDING during this time. The **IFTMiddleware2::notifyItemRegistered** callback may be used to identify when a particular source is registered with the Gallagher Controller API component, and hence identify when valid calls to **logEvent** may be made for that source.

Note that licensing problems, including a tampered or missing license file, or the absence of licensing for the Gallagher Controller API feature, will result in an FTCAPI_E_LICENSE return. There is one licensing failure, however, that will not result in an FTCAPI_E_LICENSE return: If a valid license file is found, including licensing for the Gallagher Controller API feature, but that license file does not match the license in use by the target Command Centre system, then a communications failure will result, complete with associated E_PENDING.

logEvent may only be used to log external events within the range 0 to 9. These correspond to event types “External Event 0” to “External Event 9” within the Command Centre system. Currently it is not possible to log Command Centre proprietary events through the Controller API.

The **`lEventID`** parameter may be 0 if the client middleware does not require notification of alarm acknowledgement by a Command Centre operator. Otherwise, if non-zero, the value will be included in the **IFTMiddleware2::notifyAlarmAcknowledged** callback and used to identify the event. Note that the decision about logging the event as an alarm is made by the Command Centre system according to its alarm management configuration. Refer to the ***Alarm Handling*** (on page 19) section of this specification for more details.

The **`dEventTime`** parameter will be converted by the Controller API component into an internal representation of time. This means that the parameter currently has a maximum resolution of whole seconds and a valid range of dates within the years 1970 and 2037, inclusive. Specifying dates outside that range will lead to an E_INVALIDARG return.

Given that the logged event type is configured to generate an alarm within Command Centre, the **`bHasRestoral`** parameter says whether the alarm will be ‘stateful’. A stateful alarm is displayed to operators with an ‘alarm bell’ icon indicating whether the alarm is currently active or not. Active alarms may not be processed by a Command Centre operator; the alarm condition is first required to go inactive. An event with **`bHasRestoral`** set may be notified as inactive by using the **IFTExternalEventInput2::notifyRestore** method. Refer to the ***Alarm Handling*** (on page 19) section of this specification for more details.

The **`sMessage`** and **`sDetails`** parameters will be used to describe the event occurrence anywhere in Command Centre where the event may be viewed, including the event and alarm windows and the activity report. Command Centre contains a system for displaying events in the locale language of the viewing operator. This, however, is not supported through the Gallagher Controller API and so it is the responsibility of the caller to submit the event message in an appropriate language for the target system.

The **`sMessage`** string may contain the following placeholders:

%1	Command Centre will insert the name of the event source into the event message.
%%	Command Centre will insert a single % character into the event message.

Example

```
[VC++]
HRESULT CEventSource::logEventNow(LONG lEventType, BSTR sMessage)
{
try
{
    // Get the current date/time in DATE style
    DATE dNow;
    SYSTEMTIME stNow;
    GetSystemTime(&stNow);
    SystemTimeToVariantTime(&stNow, &dNow);

    // Submit event to the Cardax FTCAPI
    HRESULT hr = m_FTCAPI ->logEvent(lEventType, 0, dNow, FALSE,
        m_mySystem -> m_bstrIdentity, m_bstrIdentity, sMessage, NULL);
    if (FAILED(hr))
        _com_issue_error(result);
…

[VB]
Private Sub logEventNow(lEventType As Long, sMessage As String)

    On Error Goto logEventNowError
    'Get the current date/time in DATE style
    Dim dNow As Date
    dNow = Now()

    'Submit event to the Cardax FTCAPI
    FTCAPI.logEvent(lEventType, 0, dNow, FALSE,
        g_sSystem, g_sSource, sMessage, vbNullString)
…
```

IFTExternalEventInput2::logCardEvent

Log an external event into the Gallagher event system.

```
[VC++]
HRESULT logCardEvent( [in] LONG lEventType, [in] LONG lEventID,
                      [in] DATE dEventTime, [in] BOOL bHasRestoral,
                      [in] LONG lCardNumber, [in] LONG lFacilityCode,
                      [in] BSTR sSystemID, [in] BSTR sItemID,
                      [in] BSTR sMessage, [in] BSTR sDetails );


[VB]
Sub logCardEvent(  lEventType As Long, lEventID As Long, _
                   dEventTime As Date, _
                   bHasRestoral As Boolean, _
                   lCardNumber As Long, _
                   lFacilityCode As Long, _
                   sSystemID As String, sItemID As String, _
                   sMessage As String, sDetails As String )
```

Parameters

`[in] LONG lEventType`

The event type code of the event occurrence. This should be within the range 0 to 9.

`[in] LONG lEventID`

A caller-defined number identifying the specific event occurrence. Assuming that the event is configured to generate an alarm within Command Centre, this identifier will be included as part of the **IFTMiddleware2::notifyAlarmAcknowledged** callback at such a time as a Command Centre operator acknowledges the alarm. Typically, this identifier will uniquely identify an event occurrence, although it is not required to. Set this parameter to 0 if you do not require notification of alarm acknowledgement.

`[in] DATE dEventTime`

The date and time of the event occurrence, in GMT. This parameter allows delayed events to be logged with their actual occurrence time. Fractions of a second will be ignored.

`[in] BOOL bHasRestoral`

Indicates whether the event is ‘stateful’ and therefore has a corresponding return-to-normal, or restoral, event.

`[in] LONG lCardNumber`

The card number of the card that generated the event. This should match a configured card within the Command Centre system.

`[in] LONG lFacilityCode`

The facility code of the card that generated the event. This should match a facility code configured against a card type in the Command Centre system.

`[in] BSTR sSystemID`

The string descriptor identifying the External System that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System Item in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sMessage`

The event message to be displayed for the event within the Command Centre event system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sDetails`

The event details string to be logged and displayed for the event within the Command Centre event system. The string can be a maximum of 128 Unicode characters.

Return Values

This method supports the same set of return values supported by **IFTExternalEventInput2::logEvent**, but with additional reasons for returning E_INVALIDARG. Refer to the documentation for that method.

E_INVALIDARG
One or more arguments are invalid, including:

- **`lCardNumber`** out of acceptable range - refer to 'Remarks', below.
- **`lFacilityCode`** out of acceptable range - refer to 'Remarks' below.

Remarks

logCardEvent is a blocking call used to log the details of a card event into the Command Centre event system. This method works in the same way as **IFTExternalEventInput2::logEvent**. Refer to the remarks for **logEvent** for detailed notes on functionality; all remarks documented for **logEvent** also apply to **logCardEvent**.

The two extra parameters in this method, **`lCardNumber`** and **`lFacilityCode`**, allow the caller to specify the card number and facility code of the card that generated the event. Given that the two fields match a configured card and card type within the Command Centre system, it is then possible to have the Command Centre event system automatically insert the name of the cardholder into the event message text.
If either of these parameters are non-zero, then the following restrictions must be met:
The **`lCardNumber`** parameter must be between 1 and 16777215.
The **`lFacilityCode`** parameter is made up of both the facility code and region code of the card type. The **`LOWORD`** contains the facility code, and must be in the range 1 to 50000. The **`HIWORD`** must contain the region code, minus character **`'A'`**, such that **`'A'`** through **`'P'`** is encoded into the range **`0x0`** through **`0xF`**.

For **logCardEvent** the **`sMessage`** string may therefore contain the following placeholders:

%**1**	Command Centre will insert the name of the event source into the event message.
%**2**	Command Centre will insert the name of the cardholder into the event message.
%%	Command Centre will insert a single % character into the event message.

An example message string may be "DURESS signalled by %2 at receiver %1".
Given that the passed **`sItemID, lCardNumber`**, and **`lFacilityCode`** all match configured items in the Command Centre system, the final displayed message text could be something like:
"DURESS signalled by Smith, John at receiver Cell 37".

Example

```
[VC++]
HRESULT CEventSource::logCardEventNow(LONG lEventType, LONG lCard, LONG lFacility,
                                                    BSTR sMessage)
{
try
{
    // Get the current date/time in DATE style
    DATE dNow;
    SYSTEMTIME stNow;
    GetSystemTime(&stNow);
    SystemTimeToVariantTime(&stNow, &dNow);

    // Submit event to the Cardax FTCAPI
    HRESULT hr = m_FTCAPI->logCardEvent(lEventType, 0, dNow, FALSE, lCard,
                                        lFacility, m_mySystem->m_bstrIdentity,
                                        m_bstrIdentity, sMessage, NULL);

    if (FAILED(hr))
        _com_issue_error(result);
…

[VB]
Private Sub logCardEventNow(lEventType As Long, lCard As Long, lFacility As Long,
_
                            sMessage As String)

    On Error Goto logCardEventNowError
    'Get the current date/time in DATE style
    Dim dNow As Date
    dNow = Now()

    'Submit event to the Cardax FTCAPI
    FTCAPI.logCardEvent(lEventType, 0, dNow, FALSE, lCard, lFacility, _
                            g_sSystem, g_sSource, sMessage, vbNullString)
…
```

IFTExternalEventInput2::notifyRestore

Notify the Command Centre event system about a condition that has returned to normal. The restore should correspond to a condition previously logged with **logEvent** or **logCardEvent**.

```
[VC]
HRESULT notifyRestore([in] LONG lEventType, [in] BSTR sSystemID,
                      [in] BSTR sItemID );

[VB]
Sub notifyRestore( lEventType As Long, sSystemID As String,
                   sItemID  As String )
```

Parameters

`[in] LONG lEventType`

The event type code of the *original* occurrence. This should be within the range 0 to 9.

`[in] BSTR sSystemID`

The string descriptor identifying the External System that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System Item in the Command Centre system. The string can be a maximum of 64 Unicode characters.

Return Values

This method supports the standard return values E_OUTOFMEMORY and E_UNEXPECTED, as well as the following:

S_OK

The operation completed successfully.

E_INVALIDARG

One or more arguments are invalid, including:

- **`lEventType`** out of the range 0 to 9.

E_PENDING

The Gallagher Controller API is not ready to accept events against the specified source. Refer to the **IFTExternalEventInput2::logEvent** documentation for a list of scenarios that could cause an E_PENDING failure.

FTCAPI_E_LICENSE

There is a problem with the Command Centre licensing. Refer to the **IFTExternalEventInput2::logEvent** documentation for a list of scenarios that could cause an FTCAPI_E_LICENSE failure.

FTCAPI_E_NETWORK

There was a network failure reaching the rest of the Command Centre system.

Remarks

notifyRestore is a blocking call used to notify the Command Centre event system about a 'stateful' event condition that has now returned to normal on a particular source.

The source item must first be registered with the Gallagher Controller API component before this call will succeed. Refer to the remarks for the **IFTExternalEventInput2::logEvent** method for details about source registration and the E_PENDING and FTCAPI_E_LICENSE returns.

If the event condition was configured to be an *alarm* within Command Centre this call will mark the alarm condition as *inactive*. When an alarm condition goes inactive in Command Centre the alarm is free to be processed and removed from the alarm window and, depending on configuration, it may also deactivate alarm relays.

notifyRestore should be called with the event code of the *original condition*, rather than the return-to-normal event.

notifyRestore does not log any event to the Command Centre event system. If the caller requires the return-to-normal event to be logged, they should submit its details through the **logEvent** or **logCardEvent** calls.

If the original condition was not logged with **`bHasRestoral`** set to TRUE, or was not configured to generate an alarm within Command Centre, then this call will have no effect.

Example

```
[VC++]
#define EVCODE_INPUT_ON   0
#define EVCODE_INPUT_OFF  1

// The source is fire input 3 on level 1
CComBSTR sSource(_T("ID_INPUT_3_LEVEL_1_FIRE"));

// Notify the return-to-normal on our source, passing the original condition
m_FTCAPI->notifyRestore(EVCODE_INPUT_ON, g_mySystem, sSource);

// Log the return-to-normal as a separate event
m_FTCAPI->logEvent(EVCODE_INPUT_OFF, 0, dNow, FALSE, g_mySystem,
                   sSource, messageRTN, NULL);
…


[VB]
Const EVCODE_INPUT_ON = 0
Const EVCODE_INPUT_OFF = 1

'The source is fire input 3 on level 1
Dim sSource As String
sSource = "ID_INPUT_3_LEVEL_1_FIRE"

'Notify the return-to-normal on our source, passing the original condition
FTCAPI.notifyRestore(EVCODE_INPUT_ON, g_mySystem, sSource)

'Log the return-to-normal as a separate event
FTCAPI.logEvent(EVCODE_INPUT_OFF, 0, dNow, FALSE, g_mySystem, sSource, _
                sMessage, vbNullString)
…
```

IFTExternalEventInput2::notifyStatus

Notify the Command Centre status system about a change in status.

```
[VC++]
HRESULT notifyStatus(  [in] BSTR sSystemID, [in] BSTR sItemID,
                       [in] LONG lState, [in] BOOL bTampered,
                       [in] BOOL bOffline, [in] BSTR sMessage );
[VB]
Sub notifyStatus( sSystemID As String, sItemID As String, _
                  lState As Long, bTampered As Boolean, _
                  bOffline As Boolean, sMessage As String )
```

Parameters

`[in] BSTR sSystemID`

The string descriptor identifying the External System for the item whose status is being updated. This should exactly match the *identification* string configured against the corresponding External System in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item whose status is being updated. This should exactly match the *identification* string configured against the corresponding External System Item in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] LONG lState`

The state of the External System Item.

`[in] BOOL bTampered`

A flag indicating whether the source is tampered.

`[in] BOOL bOffline`

A flag indicating whether the source is offline.

`[in] BSTR sMessage`

A text description of the state of the source item. This text will be displayed in the status properties page for the item within Command Centre, as well as in the Command Centre status viewer utility. Maximum of 63 Unicode characters.

Return Values

This method supports the standard return values E_OUTOFMEMORY and E_UNEXPECTED, as well as the following:

S_OK

The operation completed successfully.

E_PENDING

The Gallagher Controller API is not ready to accept events against the specified source. Refer to the **IFTExternalEventInput2::logEvent** documentation for a list of scenarios that could cause an E_PENDING failure.

FTCAPI_E_LICENSE

There is a problem with the Command Centre licensing. Refer to the **IFTExternalEventInput2::logEvent** documentation for a list of scenarios that could cause an FTCAPI_E_LICENSE failure.

FTCAPI_E_NETWORK

There was a network failure reaching the rest of the Command Centre system.

Remarks

notifyStatus is a blocking call used to notify the Command Centre status system about the current status of an External System Item (ESI).

The ESI must first be registered with the Gallagher Controller API component before this call will succeed. Refer to the remarks for the **IFTExternalEventInput2::logEvent** method for details about source registration and the E_PENDING and FTCAPI_E_LICENSE returns.

When the Gallagher Controller API component is started and an ESI is registered on the component, the client middleware should call **notifyStatus** to set the current status of the item. Later the client middleware should call **notifyStatus** whenever the status of the item changes. Client middleware may use the **IFTMiddleware2::notifySourceRegistered** callback to determine when a source is first registered with the Gallagher Controller API component.

The `lState` parameter will be used by Command Centre to decide on the icon to display against the source item. Command Centre has user-selectable icon sets; the default icon set for an External System Item is a toggle switch having three states (up, down, and central) and two overlays (screwdriver and red circle). The following table represents the relationship between the **`lState`**, **`bTampered`** and **`bOffline`** parameters and the icon displayed against the source in the Command Centre workstation:

ESI State	State of 'Toggle' icon	Overlay	State Parameter	Notes
Open / alarm / not secure / disarmed	Up	None	0	
Closed / normal / secure / armed	Down	None	1	
Offline	Central	Red circle	N/A	Determined by **`bOffline`** parameter or by Command Centre
Tamper	Central	Screwdriver	N/A	Determined by **`bTampered`** parameter

Example

```
[VC++]
const int STATUS_ALARM = 0;
const int STATUS_NORMAL = 1;

// Notify the status of the level 1 fire alarm
HRESULT hr = m_FTCAPI->notifyStatus(
            g_mySystem, CComBSTR(L"ID_INPUT_3_LEVEL_1_FIRE"),
            STATUS_ALARM, FALSE, FALSE,
            CComBSTR(L"Fire alarm ACTIVE, level one")));
…

[VB]
Const STATUS_ALARM = 0;
Const STATUS_NORMAL = 1;

'Notify the status of the level 1 fire alarm
FTCAPI.notifyStatus(g_mySystem, "ID_INPUT_3_LEVEL_1_FIRE", _
            STATUS_ALARM, FALSE, FALSE, _
            "Fire alarm ACTIVE, level one")
…
```

IFTExternalEventInput3

The **IFTExternalEventInput3** interface is inherited from the **IFTExternalEventInput2** interface. It is provided by the Gallagher Controller API component to allow external systems to submit card event and status information to the Command Centre event and status systems. This interface introduces two new methods to support long card numbers in the card events.

Method	Description
`logLongCardEvent` (on page 44)	Log an external card event into the Command Centre event system using a long card number. The card number is provided as an array of bytes and size of the array.
`logLongCardEvent2` (on page 47)	Log an external card event into the Command Centre event system using a long card number. The card number is provided as a text string along with the card number format type.

IFTExternalEventInput3::logLongCardEvent

Log an external card event into the Command Centre event system.

```
[VC++]
HRESULT logLongCardEvent( [in] LONG lEventType, [in] LONG lEventID,
                          [in] DATE dEventTime, [in] BOOL bHasRestoral,
                          [in] LONG lCardIDSize,
                          [in, size_is(lCardIDSize)]  BYTE* pCardID,
                          [in] LONG lFacilityCode,
                          [in] BSTR sSystemID, [in] BSTR sItemID,
                          [in] BSTR sMessage, [in] BSTR sDetails );

[VB]
Sub logLongCardEvent( lEventType As Long, lEventID As Long, _
                      dEventTime As Date, bHasRestoral As Boolean, _
                      lCardNumber As Long, ByRef pCardID As Byte, _
                      lFacilityCode As Long, _
                      sSystemID As String, sItemID As String, _
                      sMessage As String, sDetails As String )
```

Parameters

`[in] LONG lEventType`

The event type code of the event occurrence. This should be within the range 0 to 9.

`[in] LONG lEventID`

A caller-defined number identifying the specific event occurrence. Assuming that the event is configured to generate an alarm within Command Centre, this identifier will be included as part of the **IFTMiddleware2::notifyAlarmAcknowledged** callback at such a time as a Command Centre operator acknowledges the alarm. Typically, this identifier will uniquely identify an event occurrence, although it is not required to. Set this parameter to 0 if you do not require notification of alarm acknowledgement.

`[in] DATE dEventTime`

The date and time of the event occurrence, in GMT. This parameter allows delayed events to be logged with their actual occurrence time. Fractions of a second will be ignored.

`[in] BOOL bHasRestoral`

Indicates whether the event is 'stateful' and therefore has a corresponding return-to-normal, or restoral, event.

`[in] LONG lCardIDSize`

The size of the card number binary data, presented as an array of bytes in the pCardID parameter.

`[in, size_is (lCardIDSize) ] BYTE* pCardID`

The card number of the card that generated the event, presented as an array of bytes. The size of the array is the value of the `ICardIDSize` parameter. This should match a configured card within the Command Centre system.

`[in] LONG lFacilityCode`

The facility code of the card that generated the event. This should match a facility code configured against a card type in the Command Centre system.

`[in] BSTR sSystemID`

The string descriptor identifying the External System that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System Item in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sMessage`

The event message to be displayed for the event within the Command Centre event system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sDetails`

The event details string to be logged and displayed for the event within the Command Centre event system. The string can be a maximum of 128 Unicode characters.

Return Values

This method supports the same set of return values supported by **IFTExternalEventInput2::logCardEvent**.

Remarks

logLongCardEvent is a blocking call used to log the details of a card event into the Command Centre event system. This method works in the same way as **IFTExternalEventInput2::logCardEvent**. Refer to the remarks for **logCardEvent** for detailed notes on functionality; all remarks documented for **logCardEvent** also apply to **logLongCardEvent**.

In this method, card number is not specified as an integer. Instead, it appears in the form of a byte array. This is based on the fact that long card numbers cannot be represented by an integer (it is out of the valid range of an integer). The **`pCardID`** parameter contains the card number data after being converted to binary format. The **`lCardIDSize`** parameter specifies the length of the binary data.

For example, if the card type's Card Number Format is Text, the card number string is required to be converted to a UTF-8 encoded byte array before a long card event is logged. The following sample code shows how a Text card number string is converted to a byte array:

```
[C#]
string sCardNumber = "ABC123";
Encoding encoding = Encoding.UTF8;
byte[] bytes = encoding.GetBytes(sCardNumber);
```

IFTExternalEventInput3::logLongCardEvent2

Log an external event into the Command Centre event system.

```
[VC++]
HRESULT logLongCardEvent2( [in] LONG lEventType, [in] LONG lEventID,
                           [in] DATE dEventTime, [in] BOOL bHasRestoral,
                           [in] LONG lCardNumberFormatType,
                           [in] BSTR sCardNumber, [in] LONG lFacilityCode,
                           [in] BSTR sSystemID, [in] BSTR sItemID,
                           [in] BSTR sMessage, [in] BSTR sDetails );

[VB]
Sub logLongCardEvent2( lEventType As Long, lEventID As Long, _
                       dEventTime As Date, bHasRestoral As Boolean, _
                       lCardNumberFormatType As Long, _
                       sCardNumber As String, lFacilityCode As Long, _
                       sSystemID As String, sItemID As String, _
                       sMessage As String, sDetails As String )
```

Parameters

`[in] LONG lEventType`

The event type code of the event occurrence. This should be within the range 0 to 9.

`[in] LONG lEventID`

A caller-defined number identifying the specific event occurrence. Assuming that the event is configured to generate an alarm within Command Centre, this identifier will be included as part of the **IFTMiddleware2::notifyAlarmAcknowledged** callback at such a time as a Command Centre operator acknowledges the alarm. Typically, this identifier will uniquely identify an event occurrence, although it is not required to. Set this parameter to 0 if you do not require notification of alarm acknowledgement.

`[in] DATE dEventTime`

The date and time of the event occurrence, in GMT. This parameter allows delayed events to be logged with their actual occurrence time. Fractions of a second will be ignored.

`[in] BOOL bHasRestoral`

Indicates whether the event is ‘stateful’ and therefore has a corresponding return-to-normal, or restoral, event.

`[in] LONG lCardNumberFormatType`

The card number format.

`[in] BSTR sCardNumber`

The card number of the card that generated the event, presented as a string. This should match a configured card within the Command Centre system.

`[in] LONG lFacilityCode`

The facility code of the card that generated the event. This should match a facility code configured against a card type in the Command Centre system.

`[in] BSTR sSystemID`

The string descriptor identifying the External System that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item that is the source of the event. This should exactly match the *identification* string configured against the corresponding External System Item in the Command Centre system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sMessage`

The event message to be logged and displayed for the event within the Command Centre event system. The string can be a maximum of 64 Unicode characters.

`[in] BSTR sDetails`

The event details string to be logged and displayed for the event within the Command Centre event system. The string can be a maximum of 128 Unicode characters.

Return Values

This method supports the same set of return values supported by **IFTExternalEventInput2::logCardEvent**.

Remarks

logCardEvent2 is a blocking call used to log the details of a card event into the Command Centre event system. This method works in the same way as **IFTExternalEventInput2::logECardEvent**. Refer to the remarks for **logCardEvent** for detailed notes on functionality; all remarks documented for **logCardEvent** also apply to **logLongCardEvent2**.

In this method, card number is provided as a string rather than an integer in the **`sCardNumber`** parameter. This is based on the fact that long card numbers cannot be represented by an integer (it is out of the valid range of an integer).

The **`lCardNumberFormatType`** parameter specifies the type of the card number format. At the moment, there is only one format defined in the system, which is "Decimal" format. So the value of this parameter should always be **`1`**.

IFTMiddleware2

The **IFTMiddleware2** outgoing interface may be optionally implemented by middleware clients of the Gallagher Controller API component. It provides notification of connection and alarm acknowledgement happenings. Client middleware components connect the interface by using the standard COM mechanism for connectable objects, including the **IConnectionPoint** interface et al., provided by the Gallagher Controller API component.

Method	Description
`notifyItemRegistered` (on page 51)	Used by the Gallagher Controller API component to notify client middleware that an External System Item is known by the API and ready to have events and status submitted against it. Also used to provide the ESIs configuration string to the middleware.
`notifyItemDeregistered` (on page 53)	Used by the Gallagher Controller API component to notify client middleware that an External System Item has been deregistered from the API and may no longer have events and status submitted against it.
`notifySystemRegistered` (on page 55)	Used by the Gallagher Controller API component to notify client middleware that an External System is known by the API, and to provide the External System's configuration string to the middleware.
`notifySystemDeregistered` (on page 57)	Used by the Gallagher Controller API component to notify client middleware that an External System has been deregistered from the API.
`notifyAlarmAcknowledged` (on page 59)	Used by the Gallagher Controller API component to notify client middleware that a logged alarm has been acknowledged by an operator of the Command Centre system.

IFTMiddleware2::notifyItemRegistered

Used by the Gallagher Controller API component to notify client middleware that an External System Item is known by the API and ready to have events and status submitted against it. Also used to provide the ESIs configuration string to the middleware.

```
[VC++]
HRESULT notifyItemRegistered( [in] BSTR sSystemID, [in] BSTR sItemID,
                              [in] BSTR sConfig );

[VB]
Sub notifyItemRegistered( sSystemID As String, sItemID As String, _
                          sConfig As String )
```

Parameters

`[in] BSTR sSystemID`

The string descriptor identifying the External System for the item that has registered at the API. This is the *identification* string configured against the corresponding External System in the Command Centre system.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item that has registered at the API. This is the *identification* string configured against the corresponding External System Item in the Command Centre system.

`[in] BSTR sConfig`

The configuration string set against the item in Command Centre, in the *Configuration* section of the ESIs *Setup* property page.

Return Values

The Gallagher Controller API component ignores the return value from this method.

Remarks

notifyItemRegistered is a callback that client middleware may use to determine when event and status information may be safely submitted for particular External System Items. Middleware may also use it to get the configuration data for each External System Item.

In order for client code to receive **notifyItemRegistered** callbacks, the client must first register their implementation of **IFTMiddleware2** with the Gallagher Controller API component. This is done using the standard COM connectable objects strategy, which includes the **IConnectionPoint** interface. There are also several configuration requirements within the Command Centre system, including having a matching External Event Source item correctly configured in Command Centre, having a matching External System item correctly configured in Command Centre, and having the functioning, communicating, and licensed Command Centre system running.

Given that **notifyItemRegistered** is a notification callback, all returns from the method, including E_NOTIMPL, are ignored by the caller, the Gallagher Controller API component.

When a client has received a **notifyItemRegistered** callback for a particular ESI they should immediately call **IFTExternalEventInput2::notifyStatus** to set the current status for the source item.

Example

```
[VC++]
HRESULT CMiddleware::notifyItemRegistered(BSTR sSystemID, BSTR sItemID,
                                          BSTR sConfig)
{
    // The specified source is now registered on the API
    CESI* pESI = m_items.additem(sItemID, sConfig)
    if (pESI != NULL)
    {
        // Change the status of our item. We don't support
        // status so this func will just notify status as normal.
        pESI->setStatus(True);
    }
    return S_OK;
}

[VB]
Public Sub notifyItemRegistered(ByVal sSystemID As String, _
        ByVal sItemID As String, ByVal sConfigID As String) _
        Implements CardaxFTCAPILib.IFMiddleware2.notifyItemRegistered

    'The specified item is now registered on the API
    Call m_items.addItem(sItemID, sConfig)

End Sub
```

IFTMiddleware2::notifyItemDeregistered

Used by the Gallagher Controller API component to notify client middleware that an External System Item has been deregistered from the API and may no longer have events and status submitted against it.

```
[VC++]
HRESULT notifyItemDeregistered( [in] BSTR sSystemID, [in] BSTR sItemID,

[VB]
Sub notifyItemDeregistered( sSystemID As String, sItemID As String )
```

Parameters

`[in] BSTR sSystemID`

The string descriptor identifying the External System for the item that has been deregistered at the API. This is the *identification* string configured against the corresponding External System in the Command Centre system.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item that has been deregistered at the API. This is the *identification* string configured against the corresponding External System Item in the Command Centre system.

Return Values

The Gallagher Controller API component ignores the return value from this method.

Remarks

notifyItemDeregistered is a callback that client middleware may use to determine when an External System Item has been *deregistered* from the API instance. This is intended for use by dynamically configurable middleware, i.e. middleware that is designed to be responsive to Command Centre configuration changes without the need for restarts.

An ESI can be deregistered from an instance of Gallagher Controller API in the following scenarios:

1. If the External System Item is deleted from Command Centre.
2. If the ESI is reconfigured at Command Centre to remove it from its parent External System.
3. If the ESIs parent External System is reconfigured to remove it from its Controller or to shift it to a different Gallagher Controller API instance.

After an ESI has been deregistered from an API instance, events and status may no longer be submitted against that item at that API instance.

In order for client code to receive **notifyItemDeregistered** callbacks, the client must first register their implementation of **IFTMiddleware2** with the Gallagher Controller API component. This is done using the standard COM connectable objects strategy, which includes the **IConnectionPoint** interface.

Given that **notifyItemDeregistered** is a notification callback, all returns from the method, including E_NOTIMPL, are ignored by the caller, the Gallagher Controller API component.

IFTMiddleware2::notifySystemRegistered

Used by the Gallagher Controller API component to notify client middleware that an External System is known by the API, and to provide the External System's configuration string to the middleware.

```
[VC++]
HRESULT notifySystemRegistered( [in] BSTR sSystemID, [in] BSTR sTypeID,
                                [in] BSTR sConfig );

[VB]
Sub notifySystemRegistered( sSystemID As String, sItemID As String, _
                            sConfig As String)
```

Parameters

`[in] BSTR sSystemID`

The string descriptor identifying the External System that has been registered at the API. This is the *identification* string configured against the corresponding External System in the Command Centre system.

`[in] BSTR sTypeID`

Reserved for use by Command Centre with its FTCAPI Middleware Framework.

`[in] BSTR sConfig`

The configuration string entered for the External System in Command Centre, in the External System's *Configuration* property page.

Return Values

The Gallagher Controller API component ignores the return value from this method.

Remarks

notifySystemRegistered is a callback that client middleware may use to determine when an External System is known at the API instance. Middleware may also use it to get the configuration data for each External System.

In order for client code to receive **notifySystemRegistered** callbacks, the client must first register their implementation of **IFTMiddleware2** with the Gallagher Controller API component. This is done using the standard COM connectable objects strategy, which includes the **IConnectionPoint** interface.

If a client implements **IFTMiddleware2** they will receive a call on **notifySystemRegistered** for each External System that is assigned to the Gallagher Controller API instance. The notifcation of system registration will always happen before the notification of registration of items that connect to that system, via **notifyItemRegistered**. However, a new call to **notifySystemRegistered** will then be made whenever the system's configuration data is reconfigured at Command Centre.

Given that **notifySystemRegistered** is a notification callback, all returns from the method, including E_NOTIMPL, are ignored by the caller, the Gallagher Controller API component.

IFTMiddleware2::notifySystemDeregistered

Used by the Gallagher Controller API component to notify client middleware that an External System has been deregistered from the API.

```
[VC++]
HRESULT notifySystemDeregistered( [in] BSTR sSystemID );

[VB]
Sub notifySystemDeregistered( sSystemID As String )
```

Parameters

`[in] BSTR sSystemID`

The string descriptor identifying the External System that has been deregistered from the API. This is the *identification* string configured against the corresponding External System in the Command Centre system.

Return Values

The Gallagher Controller API component ignores the return value from this method.

Remarks

notifySystemDeregistered is a callback that client middleware may use to determine when an External System has been *deregistered* from the API instance. This is intended for use by dynamically configurable middleware - i.e. middleware that is designed to be responsive to Command Centre configuration changes without the need for restarts.

An External System can be deregistered from an instance of Gallagher Controller API in the following scenarios:

1. If the External System Item is deleted from Command Centre.
2. If the External System is reconfigured at Command Centre to remove it from its Controller or shift it to a different Gallagher Controller API instance.

The Gallagher Controller API will only make a call to **notifySystemDeregistered** after it has first called **notifyItemDeregistered** for every ESI that connects to the External System.

In order for client code to receive **notifySystemDeregistered** callbacks, the client must first register their implementation of **IFTMiddleware2** with the Gallagher Controller API component. This is done using the standard COM connectable objects strategy, which includes the **IConnectionPoint** interface.

Given that **notifySystemDeregistered** is a notification callback, all returns from the method, including E_NOTIMPL, are ignored by the caller, the Gallagher Controller API component.

IFTMiddleware2::notifyAlarmAcknowledged

Used by the Gallagher Controller API component to notify client middleware that a logged alarm has been acknowledged by an operator of the Command Centre system.

```
[VC++]
HRESULT notifyAlarmAcknowledged( [in] BSTR sSystemID, [in] LONG lEventID );

[VB]
Sub notifyAlarmAcknowledged( sSystemID As String, lEventID As Long )
```

Parameters

`[in] BSTR sSystemID`

The string descriptor identifying the External System against which the original event was logged. This is the *identification* string configured against the corresponding External System in the Command Centre system.

`[in] LONG lEventID`

The identifier of the specific alarm occurrence. This is the identifier that was specified by the caller at the time the original event was logged, via **IFTExternalEventInput2::logEvent** or **IFTExternalEventInput2::logCardEvent**.

Return Values

The Gallagher Controller API component ignores the return value from this method.

Remarks

notifyAlarmAcknowledged is a callback that client middleware may use to determine when a particular alarm occurrence has been acknowledged by an operator of the Command Centre system.

In order for client code to receive **notifyAlarmAcknowledged** callbacks, the client must first register their implementation of **IFTMiddleware** with the Gallagher Controller API component. This is done using the standard COM connectable objects strategy, which includes the **IConnectionPoint** interface.

There are several other requirements before a callback is received:

- The original event must have been successfully logged through the interface with an `lEventID` parameter not zero. Refer to documentation for **IFTExternalEventInput2::logEvent** for more details. Note that this implies the External System Item is correctly registered with the Gallagher

Controller API component and the Command Centre system is configured and working properly.

- The original event must have been configured to generate an alarm condition within Gallagher. If the event was not configured to alarm then there will be no alarm acknowledgement and hence no callback here.
- The event must have been acknowledged by the Command Centre operator.

An example application for this callback may be the automatic resetting of CCTV monitors on operator acknowledgement of an alarm.

Given that **notifyAlarmAcknowledged** is a notification callback, all returns from the method, including E_NOTIMPL, are ignored by the caller, the Gallagher Controller API component.

Example

```
[VC++]
HRESULT CMiddleware::notifyAlarmAcknowledged(BSTR sSystemID, LONG lAlarmID)
{
    // Lookup the source for this alarm
    if (m_alarmMap.find(lAlarmID) != m_alarmMap.end())
    {
        // Tell our event source proxy about the alarm ack…
        m_alarmMap[lAlarmID].notifyAlarmAck(lAlarmID);
    }
    return S_OK;
}
```

IFTTrigger2

Middleware clients of the Gallagher Controller API component may optionally implement the **IFTTrigger2** outgoing interface. It dispatches trigger strings related to Command Centre Output state changes or event occurrences. Client middleware components connect the interface by using the standard COM mechanism for connectable objects, including the **IConnectionPoint** interface et al., provided by the Gallagher Controller API component.

Method	Description
`triggerOutput` (on page 62)	Used by the Gallagher Controller API component to dispatch trigger strings related to Command Centre Output state changes.
`triggerEvent` (on page 64)	Used by the Gallagher Controller API component to dispatch trigger strings related to Command Centre event occurrence.

IFTTrigger2::triggerOutput

Used by the Gallagher Controller API component to dispatch strings triggered by Command Centre Output state changes.

```
[VC++]
HRESULT triggerOutput(  [in] BSTR sSystemID, [in] BSTR sItemID,
                        [in] LONG lState, [in] BSTR sTrigger );

[VB]
Sub triggerOutput(  sSystemID As String, sItemID As String, lState As Long, _
                    sTrigger As String )
```

Parameters

`[in] BSTR sSystemID`

The string descriptor identifying the External System for the ESI whose Output has changed state. This is the *identification* string configured against the corresponding External System in the Command Centre system.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item whose Output has changed state. This is the *identification* string configured against the corresponding External System Item in the Command Centre system.

`[in] LONG lState`

The state of the Output, where 0 is open/off and 1 is closed/on.

`[in] BSTR sTrigger`

The Output's trigger string.

Return Values

The Gallagher Controller API component ignores the return value from this method.

Remarks

triggerOutput is a callback that client middelware may use to receive strings triggered by Command Centre Output state changes. These strings are configured on the Output item's property pages, on the "Messages" tab. The string dispatched here will be the string configured against the state that the Output is entering.

In order for client code to receive **triggerOutput** callbacks, the client must first register their implementation of **IFTTrigger2** with the Gallagher Controller API component. This is done using the standard COM connectable objects strategy, which includes the **IConnectionPoint** interface.

There are also several configuration requirements within the Command Centre system, including having a matching External System Item correctly configured in Command Centre, and having the Command Centre Output (whose state is required) included in the "Trigger - Outputs" list on the External System Item property pages.

IFTTrigger2::triggerEvent

Used by the Gallagher Controller API component to dispatch strings triggered by Command Centre event occurrences.

```
[VC++]
HRESULT triggerEvent([in] BSTR sSystemID, [in] BSTR sItemID,
                           [in] BSTR sTrigger );
[VB]
Sub triggerEvent( sSystemID As String, sItemID As String,
                           sTrigger As String )
```

Parameters

`[in] BSTR sSystemID`

The string descriptor identifying the External System for the ESI that has been triggered by an event occurrence. This is the *identification* string configured against the corresponding External System in the Command Centre system.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item that has been triggered by an event occurrence. This is the *identification* string configured against the corresponding External System Item in the Command Centre system.

`[in] BSTR sTrigger`

The ESIs event trigger string, as defined on the ESIs **Trigger - Event** property page.

Return Values

The Gallagher Controller API component ignores the return value from this method.

Remarks

triggerEvent is a callback that client middleware may use to receive user-configured strings triggered by Command Centre event occurrence. These strings are configured on the External System Item's "Trigger - Event" property page.

In order for client code to receive **triggerEvent** callbacks, the client must first register their implementation of **IFTTrigger2** with the Gallagher Controller API component. This is done using the standard COM connectable objects strategy, which includes the **IConnectionPoint** interface.

There are also several configuration requirements within the Command Centre system, including having a matching External System Item correctly configured in Command Centre, and having the External System Item assigned to a "Cameras, Macros, and External System Items" list on an Action Plan. Then, whenever the related Command Centre Action Plan fires, each of the assigned External System Items will be triggered and their configured event trigger strings sent out through this API method.

The trigger string dispatched here will have all of its configured parameters (event details and control characters) instantiated before dispatch, except those parameters that have no value, like *cardholder name* in a non-card event.

IFTMessages

Middleware clients of the Gallagher Controller API component may optionally implement the **IFTMessages** outgoing interface. It dispatches display messages that are similar to informational and feedback messages as appear on the IDT and other readers with a text display, (e.g. "Please Enter" and the "Secure" prompts). Client middleware components connect the interface by using the standard COM mechanism for connectable objects, including the **IConnectionPoint** interface et al., provided by the Gallagher Controller API component.

Method	Description
`displayMessage` (on page 67)	Used by the Gallagher Controller API component to dispatch access response messages related to card badging and door access.

IFTMessages::displayMessage

Used by the Gallagher Controller API component to dispatch access response messages.

```
[VC++]
HRESULT displayMessage ( [in] BSTR sSystemID, [in] BSTR sItemID,
                         [in] LONG lMessageID, [in] BOOL bClearDisplay,
                         [in] SAFEARRAY (BSTR) params );
```

```
[VB]
Sub displayMessage (  sSystemID As String, sItemID As String, _
                      lMessageID As Integer, bClearDisplay As Integer, _
                      params() As String)
```

Parameters

`[in] BSTR sSystemID`

The string descriptor identifying the External System for the ESI that has been triggered by an event occurrence. This is the *identification* string configured against the corresponding External System in the Command Centre system.

`[in] BSTR sItemID`

The string descriptor identifying the External System Item that has been triggered by an event occurrence. This is the *identification* string configured against the corresponding External System Item in the Command Centre system.

`[in] LONG lMessageID`

Indicates the type of message to display, for example "Access Granted".

`[in] BOOL bClearDisplay`

When true this flag indicates that the message supersedes any previously displayed message. When false, the message should be overlaid on top of the previous message.

`[in] SAFEARRAY params`

Array of parameter value strings, such as cardholder name, competency name, competency message, or expiry date. The meaning of each value is this array depends on the value of lMessageID.

Return Values

The Gallagher Controller API component ignores the return value from this method.

Remarks

displayMessage is a callback that client middleware may use to receive display messages that usually appear on the IDT and other readers with a text display. Only External System Items used in place of normal card readers can receive display messages, (e.g. External System Items assigned as entry readers or exit readers on the properties of a Door item in Command Centre.

In order for client code to receive **displayMessage** callbacks, the client must first register their implementation of **IFTMessages** with the Gallagher Controller API component. This is done using the standard COM connectable objects strategy, which includes the **IConnectionPoint** interface.

There are also several configuration requirements within the Command Centre system, including having a matching External System Item correctly configured in Command Centre, and having the External System Item used in a place of a normal card reader in Command Centre, (e.g. assigned to a door as an entry or exit reader).

The following table contains message types:

Description/IDT Message	**Value**	**Param0**	**Param1**	**Param2**	**Param3**
Access granted/Please enter	0	Cardholder			
Timeout/Too slow*	1				
Wrong PIN	2	Cardholder			
Unknown card!	3				
Re-issued card	4				
PIN please*	5	Cardholder			
No access!	6	Cardholder			
No access at <time>	7	Cardholder			
Not an escort!	8				
Invalid facility code/Not one of ours!	9				
Same card used twice for dual-authority/Same card	10				
Passback attempt	11				
Secure	12				
Free	13				
Code only mode/Code:	14				
Dual-authority	15				

Description/IDT Message	Value	Param0	Param1	Param2	Param3
Alarms set	16				
Escort please	17				
Waiting for 2nd authority	18				
Access override*	19				
Invalid key*	20				
One visitor at a time for this door!	21				
No authority to override alarms!	22				
No authority to override access!	23				
Dual-authority invalid!	24				
Standby...	25				
Failed to set alarms	26				
Card please*	27				
Invalid code	28				
Failed to unset alarms	29				
Setting alarms	30				
Unsetting alarms	31				
Alarm state unknown. Please try again*	32				
Alarms cannot be set in free*	33				
Cannot override access while alarms are set*	34				
<name> Already registered*	35	Cardholder			
Please enter	36	Cardholder			
Usercode please*	37	Cardholder			
Wrong usercode*	38	Cardholder			
Lockdown	39				
[Access denied] due to lockdown	40				
Lckdwn*	41				
Competency missing	42	Cardholder	Competency	Message	
Competency expired	43	Cardholder	Competency	Message	Expired Date
Competency expiry due	44	Cardholder	Competency	Message	Expiry Due Date

* These messages are never sent to an external display.

The integers contained in the second column are possible values of the **`lMessageID`** parameter. The last four columns contain the parameters corresponding to each message type. Different types of messages may have a different number of parameters. For example, the "Access Granted" message (it's ID is 0), has one parameter, which is the name of the cardholder who is given access to a door. Some messages do not have any parameters such as the "Unknown card!" message. The competency related messages, (i.e. Competency Missing, Competency Expired and Competency Expiry Due) have up to four parameters which are cardholder name, competency name, competency message that is configurable in Command Centre, and expiry due date (or expired date).

Where possible, for messages with similar parameters the parameters appear in the same locations in the params array. For example, all access response messages for which a cardholder name exists, have the cardholder name first, and for competency related messages this is followed by the name of the competency and then the message configured against the competency in Command Centre (including placeholders for other parameters). Parameters that are not always present come later, (e.g. expiry due date).

Dates are represented using POSIX time format of: [[CC]YY]MMDDhhmm[.SS] and have already been converted into the Controller's local time zone.