SLMessageServer SDK – оглавление

SLMessageServer SDK – краткое описание

Пакет разработчика программного обеспечения SLMessageServer SDK предназначен для написания приложений, управляющих программой FDOnAir и/или принимающих сообщения от программы FDOnAir.

Для обмена сообщениями используется COM-сервер SLMessageServer, включающий два COM-обекта: SLMSConnection для организации связи и SLMSQueue для приема сообщений. Описание этих объектов и их основных интерфейсов приведено в данном документе.

Приложение FDOnAir использует сервер SLMessageServer для посылки и приема команд. Описание команд также приведено в данном документе.

Как создавать проекты с использованием SLMessageServer SDK

Программные модули SLMessageServer SDK представляют собой COM-объекты, устанавливаемые и регистрируемые в системе при установке ПО плат FD300/FDExt. Поскольку для вызова объектов используется технология COM, в начале работы вызывающее приложение обязано вызвать функцию CoInitialize(NULL), а в начале исполнения каждого из рабочих потоков (если они используются) CoInitializeEx(NULL, COINIT_MULTITHREADED), в противном случае любой последующий вызов (и даже создание экземпляра объекта) вернет код ошибки «не инициализирована библиотека COM». Соответственно, при завершении работы каждого потока необходимо вызывать CoUnitialize().

Примечание: в главном потоке необходимо вызвать в начале работы (обычно в процедуре инициализиции класса приложения) CoInitialize(NULL). CoUninitialize() при выходе из программы можно не вызывать.

Для работы с FDOnAirMirroring SDK необходимо импортировать в разрабатываемое приложение классы из ***SLMessageServer 1.0 Type Library*** стандартным для COM или OLE объектов образом.

Либо можно вставить в файл 'stdafx.h' следующие строки (комментарий добавлен для удобства):

```
// SLMessageServer
#import "libid:80418EB6-3E79-4F86-AB75-9549D4F38BD8" named_guids raw_interfaces_only
using namespace SLMessageQueue2Lib;
```

Примечание: компилятор может выдать предупреждение о том, что названия процедур “GetMessage” и “SendMessage” уже определены. Можно не обращать внимания на это предупреждение, но лучше переименовать методы, вставив в файл 'stdafx.h' следующее:

```
// SLMessageServer
#import "libid:80418EB6-3E79-4F86-AB75-9549D4F38BD8" named_guids raw_interfaces_only\
    rename("GetMessage", "SLGetMessage")\
    rename("SendMessage", "SLSendMessage")
using namespace SLMessageQueue2Lib;
```

Далее в своей программе нужно из интерфейсов вызывать методы с названиями SLSendMessage и SLGetMessage соответственно (см. ниже описание интерфейсов ISLMSConnection и ISLMSQueue в разделе «SLMessageServer SDK – интерфейсы»).

SLMessageServer SDK – объекты

В данном разделе перечислены все основные объекты, предоставляемые SLMessageServer SDK. Для каждого объекта указано, какие интерфейсы он предоставляет. Сами интерфейсы и подробное описание их методов приведены в разделе «SLMessageServer SDK – интерфейсы».

```
coclass SLMSConnection
{
interface ISLMSConnection;
};
```

Объект **SLMSConnection** предназначен для инициализации сервера, предназначенного для обмена сообщениями между приложениями на одной или на разных машинах.

```
coclass SLMSQueue
{
interface ISLMSQueue;
};
```

Объект **SLMSQueue** предназначен для приема сообщений от внешний приложений.

SLMessageServer SDK – интерфейсы

ISLMSConnection

CLSID класса объекта: CLSID_SLMSConnection

Данный интерфейс предназначен для организации связи с другими приложениями.

HRESULT GetName([out] BSTR* Name) – возвращает имя текущей машины.

Примечание: при работе только на локальном компьютере (не через сеть) данный метод можно не вызывать, а использовать в качестве имени компьютера строку «localhost» (без кавычек).

HRESULT CreateQueue([out] SLMSErrorCode* ErrorCode, [out] HRESULT* ResultCode, [in] BSTR Name, [in] long NotifyHwnd, [in] long NotifyCmd, [in] long NotifyWParam, [in] long NotifyLParam, [out] IUnknown** retQueue) – создает очередь для приема сообщений с именем Name. При поступлении в очередь сообщения сервер посылает окну NotifyHwnd сообщение NotifyCmd с параметрами WPARAM = NotifyWParam и LPARAM = NotifyLParam. Из возвращаемого параметра retQueue нужно получить интерфейс ISLMSQueue (см. ниже).

ВНИМАНИЕ: имя очереди должно быть уникальным! На одном компьютере не допускается создавать две и более очереди с одним и тем же названием. Программа FDOnAir при запуске создает очередь с названием «FDOnAir1» для экземпляра с номером 1. Настоятельно рекомендуется в других программах создавать очереди с названием, вообще не содержащим строки «FDOnAir», чтобы не вышло пересечения с очередью сообщений какого-либо экземпляра программы FDOnAir. Если попытаться создать очередь с названием, под которым уже существует какая-либо очередь, то функция CreateQueue вернет ошибку.

HRESULT SendMessage([out] SLMSErrorCode* ErrorCode, [out] HRESULT* ResultCode, [in] long MessId, [in] long ReplyTo, [in] BSTR ToMachine, [in] BSTR ToQueue, [in] BSTR Subject, [in] BSTR Body) – посылает сообщение (команду) в очередь ToQueue на машине ToMachine (на локальном компьютере в качестве имени машины лучше использовать строку «localhost», без кавычек). Параметры сообщения:

MessId – идентификатор сообщения (произвольное число). Этот идентификатор может использоваться, например, для получения подтверждения об успешном выполнении данной команды.

ReplyTo – идентификатор сообщения, в ответ на которое было создано это сообщение.

Subject – некоторая строка, влияющая на распознавание команды. Например, программа FDOnAir для зеркалирования использует строку "OnAir1.Mirror".

Body – собственно команда.

ISLMSQueue

CLSID класса объекта: CLSID_SLMSQueue

Данный интерфейс обеспечивает работу с очередью сообщений – прием сообщений от внешних приложений и посылку сообщений с возможностью получения подтверждения о доставке. Создается очередь с помощью метода CreateQueue интерфейса ISLMSConnection (см. выше).

HRESULT GetName([out] BSTR* Name) – возвращает имя очереди.

ВНИМАНИЕ: имя очереди должно быть уникальным! На одном компьютере не допускается создавать две и более очереди с одним и тем же названием. Это нужно иметь ввиду при создании очереди (см. выше). Если попытаться создать очередь с названием, под которым уже существует какая-либо очередь, то функция CreateQueue вернет ошибку.

HRESULT GetCounts([out] long* Count, [out] long* Lost) – возвращает в параметре Count количество сообщений в очереди, а в параметре Lost – количество сообщений, потерянных с момента создания очереди.

HRESULT GetMessage([out] SLMSErrorCode* ErrorCode, [out] HRESULT* ResultCode, [out] SLMSMessageType* MessType, [out] long* MessId, [out] long* ReplyTo, [out] long* SentLo, [out] long* SentHi, [out] BSTR* FromMachine, [out] BSTR* FromQueue, [out] BSTR* Subject, [out] BSTR* Body) – вынимает из очереди очередное сообщение. В параметре MessType возвращается тип сообщения (см. примечание 1):

mtIgnore – тестовое сообщение, обрабатывать не требуется.

mtMessage – сообщение от другого приложения.

mtError – сообщение об ошибке при доставке посланного сообщения.

mtNotify – сообщение об успешной доставке сообщения, посланного с флагом Notify

В параметрах SentLo и SentHi возвращается структура FILETIME (64 бита) с системным временем посылания сообщения. В параметрах FromMachine и FromQueue указывается имя машины и имя очереди, от которых пришло сообщение. В поле Body содержится текст самой команды.

HRESULT SendMessage([out] SLMSErrorCode* ErrorCode, [out] HRESULT* ResultCode, [in] long MessId, [in] long ReplyTo, [in] long Notify, [in] BSTR ToMachine, [in] BSTR ToQueue, [in] BSTR Subject, [in] BSTR Body) – команда аналогична команде SendMessage интерфейса ISLMSConnection за одним исключением: если в параметре Notify указано ненулевое значение, то в ответ обязательно придет сообщение об успешной доставке посланного сообщения или об ошибке.

HRESULT Ping([out] SLMSErrorCode* ErrorCode, [out] HRESULT* ResultCode, [in] long MessId, [in] BSTR ToMachine, [in] BSTR ToQueue) – послать тестовое сообщение.

Примечания:

1. ВНИМАНИЕ: в качестве имени компьютера в параметре ToMachine нельзя передавать пустую строку! Если это локальный компьютер, нужно передавать строку "localhost", либо получить имя локального компьютера с помощью вызова GetName из ISLMSConnection. Для удаленного компьютера нужно указать его сетевое имя (например, "Server2") либо IP-адрес (например, "193.125.41.123").

2. ВНИМАНИЕ: имя очереди должно быть уникальным! На одном компьютере не допускается создавать две и более очереди с одним и тем же названием. Программа FDOnAir при запуске создает очередь с названием «FDOnAir1» для экземпляра с номером 1. Настоятельно рекомендуется в других программах создавать очередь с названием, вообще не содержащим строки «FDOnAir», чтобы не вышло пересечения с очередью сообщений какого-либо экземпляра программы FDOnAir. Если попытаться создать очередь с названием, под которым уже существует какая-либо очередь, то функция CreateQueue вернет ошибку.

3. Если сообщение на другой компьютер не доходит (и/или не приходит сообщение в ответ), попробуйте отключить Брандмауэр (Firewall) системы Windows на обоих компьютерах, либо настроить в нем исключения (см. ниже раздел "Настройка Брандмауэра (Firewall) Windows").

4. В ответ на посылку сообщения сервер SLMessageServer может послать в очередь одно или два сообщения. В случае возникновения ошибки при передаче сообщения от машины к машине или при постановке в очередь сервер сформирует сообщение с типом mtError, при этом в параметре ReplayTo будет указан идентификатор посланного сообщения (его MessId). Если при отправке сообщения был установлен флаг Notify, то в случае отсутствия ошибки сервер сформирует одно или два сообщения с типом mtNotify. При этом в параметре ReplayTo также будет указан идентификатор посланного сообщения (его MessId). Первое сообщение может быть послано при передаче на другую машину (в этом случае параметр FromQueue будет пустым). Второе сообщение посылается при успешном извлечении адресатом сообщения из очереди. В обоих случаях в полях Subject и Body возвращаются поля ToMachine и ToQueue исходного сообщения.

5. То, что вызов SendMessage из ISLMSConnection или ISLMSQueue вернул S_OK, еще не означает, что сообщение успешно отправлено и доставлено. Это означает, что сообщение успешно положено в очередь на отправку. Истинный результат выполнения можно узнать из параметра ErrorCode.

   Во всех выше перечисленных методах возвращается две ошибки: SLMSErrorCode ErrorCode и HRESULT ResultCode. В параметре ResultCode возвращается стандартный результат выполнения COM-запроса. Там всегда должно быть S_OK. В параметре ErrorCode может быть одно из следующих значений:

   mcUnknown – ошибка COM, конкретный код ошибки см. в ResultCode.

   mcSuccess – все хорошо.

   mcNoMachine – нет указанной машины (см. ResultCode).

   mcMachineError – ошибка при обращении к указанной машине (например, потеря связи, см. ResultCode).

   mcNoQueue – нет указанной очереди.

   mcQueueError – ошибка при обращении к указанной очереди (например, очередь переполнена).

   mcNoMessage – нет никакого сообщения.

mcQueueExist – очередь найдена и работает нормально (возвращается при посылке тестового сообщения).

Настройка Брандмауэра (Firewall) Windows

Прежде чем обмениваться сообщениями с другим компьютером посредством SLMessageServer, необходимо либо отключить Windows Firewall (Брандмауэр) на обоих компьютерах, либо настроить исключения. Совсем отключать Firewall не рекомендуется из соображений безопасности, поэтому можно отключить его лишь на время начальных экспериментов, убедиться, что обмен сообщениями работает, а затем настроить исключения на каждом компьютере, который участвует в обмене сообщениями. Фактически, нужно разрешить прием сообщений серверу SLMessageServer с других компьютеров.

Для этого нужно проделать следующие действия:

1. Выйти из всех экземпляров программ, который обмениваются сообщениями (FDOnAir, ваша программа для отправки/приема сообщений).
2. Вызвать команду «Пуск->Панель управления->Брандмауэр Windows».
3. В первой закладке «Общие» выбрать «Включить», галочку «Не разрешать исключения» отключить.
4. Во второй закладке «Исключения» попробовать найти в списке программ «SLMessageServer2». Если она там уже есть, то включить галочку слева от нее (или убедиться, что она включена).
5. Если программы в списке нет, то нужно нажать кнопку «Добавить программу», нажать кнопку «Обзор», указать программу:

   C:\Program Files\Common Files\SoftLab-Nsk\SLMessageServer2.exe

   (вместо диска 'C:' может быть другой, куда установлено ПО ForwardT Software)
6. При необходимости можно нажать кнопку «Изменить область» и задать, с каких компьютеров можно разрешать соединение: любых, только локальной подсети, либо задать конкретный список компьютеров (IP-адресов). После задания области нажать OK.
7. Нажать OK. Убедиться, что программа «SLMessageServer2.exe» добавилась в список, и галочка слева от нее включена. Нажать OK (закрыть настройки Брандмауэра Windows).
8. Все эти действия (пп. 1-7) проделать на каждом компьютере, который участвует в обмене сообщениями через SLMessageServer.

После этого обмен сообщениями должен работать, при условии, что при отправке сообщений правильно указано имя другого компьютера, имя очереди для приема сообщений на нем, и запущена программа, которая создает эту очередь (например, FDOnAir).

Работа с приложением FDOnAir

Теперь перейдем к описанию того, как приложение FDOnAir использует сервер SLMessageServer для посылки и приема команд.

Каждый экземпляр программы FDOnAir #1, FDOnAir #2 и т.д. открывает для приема очередь с именем "FDOnAir1", "FDOnAir2" и т.д. соответственно.

В установках программы указывается, исполняет ли программа получаемые извне команды. Отдельно указывается, посылает ли программа команды при действиях оператора, и на какую машину. При этом программа посылает на указанную машину сообщения в очередь с именем своей очереди.

Поле сообщения "Subject" всегда должно содержать строку "OnAir1.Mirror". Цифра 1 в строке указывает версию протокола зеркалирования и никак не связана с номером экземпляра программа FDOnAir.

Поле сообщения "Body" содержит строку конкретной команды (список команд приводится ниже).

Исполняемые строки, за исключением некоторых команд, всегда пишутся в лог-файл.

В лог-файл пишется сначала источник команды (интерактивная кнопка, горячая клавиша или сообщение от внешнего приложения), а затем сама строка.

Строка состоит из имени команды и параметров, разделенных пробелами.

В именах команд большие и маленькие буквы не различаются.

Список команд, поддерживаемых программой FDOnAir

`Loading.Break`
Прервать загрузку расписания (состояние расписания на зеркалящихся машинах может оказаться разным)

`Player.SetVolume Node Value`
Установить громкость звуковой линии в децибелах
Например: "Player.SetVolume 1 -48"
Не пишется в Лог-файл.

`Player.SetForeFade Duration`
Установить длительность проявления/гашения логотипа/титров в миллисекундах
Например: "Player.SetForeFade 500"
Не пишется в Лог-файл.

`Player.SetBackFade Duration`
Установить длительность перехода для команд фона в миллисекундах
Например: "Player.SetBackFade 500"
Не пишется в Лог-файл.

`Player.PrepareVideo Number`
Подготовить видео на проход (превью)
Например: "Player.PrepareVideo 1"

`Player.Video Number`
Включить видео на проход
Например: "Player.Video 1"
"Player.Video 0" - включить на проход ранее подготовленное видео

`Player.SwitchVideo Number`
Переключить видео, идущее на проход
Например: "Player.SwitchVideo 1"
Команда не останавливает работу расписания.

Player.PureVideo Number
Включить видео на проход с выключением логотипа и титров
Например: "Player.PureVideo 1"

Player.Default Number
Показать заставку
Например: "Player.Default 1"

Player.SetLogotip 0/1
Разрешить/запретить показ логотипа
Например: "Player.SetLogotip 1"

Player.SetTitling 0/1
Разрешить/запретить показ титров
Например: "Player.SetTitling 1"

Player.TurnLogotip
Изменить состояние показа логотипа

Player.TurnTitling
Изменить состояние показа титров

Player.SoundFileStop
Остановить проигрыш звукового файла

Player.ForeTitleStop
Остановить проигрыш полноэкранного титра

Player.SetAux 0/1
Включить/выключить микрофон
Например: "Player.SetAux 1"

Player.TurnAux
Изменить состояние микрофона

Player.SetMute Node 0/1
Установить заглушение звуковой линии
Например: "Player.SetMute 1"

Player.SetAudioBound 0/1/2
Установить диапазон индикаторов звукового миксера -48/-72/-96 децибел
Например: "Player.SetAudioBound 3"

Player.SetTitleButton Number 0/1
Установить состояние тировальной кнопки
Например: "Player.SetTitleButton 1 0"

Player.AbortTitleButton Number
Прервать задание тировальной кнопки
Например: "Player.AbortTitleButton 1"

Player.ClearTitleButton Number
Очистить задание тировальной кнопки
Например: "Player.ClearTitleButton 1"

Player.ReloadTitleButton Number
Перезагрузить задание тировальной кнопки
Например: "Player.ReloadTitleButton 1"

Player.SetTitleObjectTask {ObjectName} TaskName
Загрузить задание в тировальный объект
Например: "Player.SetTitleObjectTask {Строка1} D:/реклама/утренний блок.spt"

Player.SetWaitVideo 0/1
Установить флаг – должно ли расписание ожидать окончания исполнения текущей команды видео
Например: "Player.SetWaitVideo 0"

Player.SetWaitSound 0/1
Установить флаг – должно ли расписание ожидать окончания исполнения текущего файла звука
Например: "Player.SetWaitSound 0"

Player.SetWaitTitle 0/1
Установить флаг – должно ли расписание ожидать окончания исполнения текущего полноэкранного титра
Например: "Player.SetWaitTitle 0"

Player.SetWaitPause 0/1
Установить флаг – должно ли расписание ожидать окончания исполнения текущей команды индикатора "Пауза"
Например: "Player.SetWaitPause 0"

Shedule.Start
Запустить расписание

Shedule.ShowFirst
Показать первый кадр

Shedule.Break
Прервать проигрыш

Shedule.SetPause 0/1
Продолжить исполнение/остановить расписание
Например: "Shedule.SetPause 0"

Shedule.SetRunShedule 0/1
Выбрать текущее исполняемое расписание
Например: "Shedule.SetRunShedule 1"

Shedule.TurnRunShedule
Сменить текущее исполняемое расписание

Shedule.SetEditPosition Shedule Line
Задать позицию редактирования (первая строка в расписании имеет индекс 0)
Например: "Shedule.SetEditPosition 0 5" – верхнее расписание, 6-ая строка

Shedule.SetRunPosition Shedule Line
Задать позицию исполнения (первая строка в расписании имеет индекс 0)
Например: "Shedule.SetRunPosition 1 10" – нижнее расписание, 11-ая строка

Shedule.SetItemDuration Shedule Line Duration
Задать длительность команды в расписании в миллисекундах
Например: "Shedule.SetItemDuration 0 5 1500"

Shedule.SetItemFade Shedule Line Duration
Задать длительность перехода команды в расписании в миллисекундах
Например: "Shedule.SetItemFade 0 5 100"

Shedule.SetItemWaitDecrease Shedule Line Duration
Задать параметр WaitDecrease для команд ожидания в расписании в миллисекундах
Например: "Shedule.SetItemWaitDecrease 0 5 1500"

Shedule.SetItemName Shedule Line Name
Задать имя текущей команды в расписании (для команд с редактируемым именем, например, начало блока)
Например: "Shedule.SetItemName 0 5 Именованный блок"

Shedule.SetItemWaitType Shedule Line 0/1
Задать переключение команд "wait operator", "wait follow" или "wait time" – пассивное/активное состояние
Например: "Shedule.SetItemWaitType 0 5 0"

Shedule.SetItemWaitStart Shedule Line StartTime
Задать время старта команды "wait time" в миллисекундах
Например: "Shedule.SetItemWaitStart 0 5 72000000"

Shedule.SetItemInOut Shedule Line In Out

Задать параметры In и Out для команды в расписании в миллисекундах
Например: "Shedule.SetItemInOut 0 5 1000 60000"

Shedule.SetItemWaitDuration Shedule Line 0/1

Установить флаг – должно ли расписание ожидать окончания исполнения команды (или должно сразу запустить следующую команду)
Например: "Shedule.SetItemWaitDuration 0 5 0"

Shedule.SetItemLogo Shedule Line 0/1/2

Задать Logo-флаг команды: None/Off/On
Например: "Shedule.SetItemLogo 0 5 2"

Shedule.SetItemTitle Shedule Line 0/1/2

Задать Title-флаг команды: None/Off/On
Например: "Shedule.SetItemTitle 0 5 2"

Shedule.AddItem Shedule Line Insert Logo Title Command

Добавить в расписание команду:
Insert - 0/1 задает моду Замена/Вставка
Logo и Title - 0/1/2 задают соответствующие флаги
Command - строка команды, как в файле расписания
Например: "Shedule.AddItem 0 5 1 0 0 movie 0:00:02.00 [2.00] D:\TEMP\Glass.avi"

Shedule.SetInsertMode 0/1

Задает моду Замена/Вставка
Например: "Shedule.SetInsertMode 1"

Shedule.SetLockRun 0/1

Задает удержание строки исполнения в зоне видимости
Например: "Shedule.SetLockRun 0"

Shedule.DeleteItem Shedule Line

Удалить команду из расписания
Например: "Shedule.DeleteItem 0 5"

Shedule.DeleteItemAbove Shedule Line

Удалить из расписания команды выше указанной позиции
Например: "Shedule.DeleteItemAbove 0 5"

Shedule.EraseList Shedule

Очистить расписание
Например: "Shedule.EraseList 0"

Shedule.SaveToFile Shedule FileName

Сохранить расписание в файл
Например: "Shedule.SaveToFile 0 D:/My Documents/example.air"

Shedule.CheckState

Проверить состояние команд в расписаниях

Shedule.TimeShift Shedule Line Shift

Сдвинуть все команды ожидания времени, начиная с указанной линии, на время в миллисекундах
Например: "Shedule.TimeShift 1 1 -3600000"

Shedule.ReadFromFile Shedule FileName

Загрузить расписание из файла
Например: "Shedule.ReadFromFile 1 D:\TEMP\g.air"

Shedule.InsertFromFile Shedule Line FileName

Вставить расписание из файла
Например: "Shedule.InsertFromFile 1 5 D:\TEMP\g.air"

Shedule.SkipBlock

Перейти к исполнению следующего блока

Примечания:

Поскольку практически все команды записываются в лог-файл, общая рекомендация для четкого понимания формата команды выглядит так: запустите программу FDOnAir и в ней выполните те действия, которые Вы хотите выполнить из внешнего приложения. То, что будет записано программой FDOnAir в лог-файл, и есть та последовательность команд, которую Вы должны послать из внешнего приложения.

Опрос состояния приложения FDOnAir

Сервер SLMessageServer можно использовать и для получения информации о текущем состоянии приложения FDOnAir. Для получения конкретной информации необходимо послать соответствующий запрос. В ответ на этот запрос приложение FDOnAir вышлет ответное сообщение с требуемой информацией.

Основное отличие сообщения-запроса от сообщения-команды – другое текст в поле Subject. Поле сообщения "Subject" для сообщения-команды всегда должно содержать строку "OnAir1.Mirror", а для сообщения-запроса – строку "OnAir1.MirrorQuery". Поле сообщения "Body" содержит список запрашиваемых параметров, разделенных переводом строки.

В ответ приложение OnAir посылает сообщение с полем "Subject", содержащем строку "OnAir1.MirrorAnswer". Поле сообщения "Body" содержит тот же список параметров, дополненных требуемой информацией. Нераспознанные строки возвращаются без изменений (например, их можно использовать как разделитель или комментарий).

Список запросов, поддерживаемых программой FDOnAir

Ниже приведен перечень запросов, поддерживаемых приложением FDOnAir. Собственно текст строки-запроса указан до вопросительного знака. Ответ включает дополнительные параметры, описанные поле вопросительного знака. Сам вопросительный знак не должен передаваться – он служит только для разделения параметров в описании запросов.

`Player.Volume Node ? Value`
Громкость звуковой линии в децибелах
Например: "Player.Volume 1 -48"

`Player.Mute Node ? 0/1`
Заглушение звуковой линии
Например: "Player.Mute 1 0"

`Player.Peak Node ? Value`
Текущее пиковое значение звуковой линии в децибелах
Например: "Player.Peak 1 -13"

`Player.AudioBound 0/1/2`
Диапазон индикаторов звукового миксера -48/-72/-96 децибел
Например: "Player.AudioBound 3"

`Player.ForeFade ? Duration`
Длительность проявления/гашения логотипа/титров в миллисекундах
Например: "Player.ForeFade 500"

`Player.BackFade ? Duration`
Длительность перехода для команд фона в миллисекундах
Например: "Player.BackFade 500"

Player.Logotip ? 0/1
Разрешение показа логотипа
Например: "Player.Logotip 1"

Player.Titling ? 0/1
Разрешение показа титров
Например: "Player.Titling 1"

Player.Aux ? 0/1
Включение микрофона
Например: "Player.Aux 1"

Player.Video ? StartTime PlayedTime Wait(0/1) Command
Текущая команда Video
StartTime - время старта команды
PlayedTime - длительность проигрывания команды
Wait(0/1) - должно ли расписание ждать окончания проигрыша
Command - строка команды, как в файле расписания
Например: "Player.Video 14:03:11:00 0:00:01.45 1 movie 0:00:02.00 [2.00] D:\TEMP\Glass.avi"
Если PlayedTime >= Duration, то проигрыш закончен
Если индикатор пуст, строка команды отсутствует

Player.Sound ? StartTime PlayedTime Wait Command
Текущая команда Sound

Player.Title ? StartTime PlayedTime Wait Command
Текущая команда полноэкранных титров

Player.Pause ? StartTime PlayedTime Wait Command
Текущая команда индикатора Pause

Player.TitleButton Number ? {ObjectName} On Run LoopCount RestTime Duration TaskName
Информация о титровальной кнопке
Number - номер кнопки, 0 для логотипа
ObjectName - имя назначенного титровального объекта
On(0/1) - включена ли кнопка
Run(0/1) - проигрывается ли объект (необходимо On & Player.Titling)
LoopCount - количество повторов (-1 для зацикленных)
RestTime - сколько осталось проиграть
Duration - длительность задания
TaskName - имя задания
Например: "Player.TitleButton 1 {Строка1} 1 0 -1 0:01:30.62 0:02:00.00 D:/реклама/утренний блок.spt"

Player.TitleObject {ObjectName} ? On Run LoopCount RestTime Duration TaskName
Информация о титровальном объекте
Например: "Player.TitleObject {Строка1} 1 0 -1 0:01:30.62 0:02:00.00 D:/реклама/утренний блок.spt"

Player.TitleObjectList ? Count {ObjectName1} {ObjectName2} ...
Список титровальных объектов
Например: "Player.TitleObjectList 3 {Логотип} {Строка1} {Строка2}"

Shedule.Pause ? 0/1
Остановлено ли расписание
Например: "Shedule.Pause 1"

Shedule.Length Shedule ? Lenght
Количество строк в расписании
Например: "Shedule.Length 0 257"

Shedule.EditPosition ? Shedule Line
Позиция редактирования
Например: "Shedule.EditPosition 0 12"

Shedule.ActiveShedule ? Shedule

Активное расписание 0/1
Например: "Shedule.ActiveShedule 1"

Shedule.RunPosition Shedule ? Line

Позиция исполнения в расписании
Например: "Shedule.RunPosition 1 12"

Shedule.InsertMode ? 0/1

Мода "Замена"/"Вставка"
Например: "Shedule.InsertMode 1"

Shedule.LockRun ? 0/1

Удержание строки исполнения в зоне видимости
Например: "Shedule.LockRun 1"

Shedule.Item Shedule Line ? Logo Title Command

Команда из расписания
Logo, Title - 0/1/2 задает соответствующие флаги 0-None/1-Off/2-On
Command - строка команды, как в файле расписания
Например: "Shedule.Item 0 5 0 0 movie 0:00:02.00 [2.00] D:\TEMP\Glass.avi"

Shedule.ItemEx Shedule Line ? State Date Time Logo Title Command

Команда из расписания с состоянием и временем старта
State - None(0), Loading(1), GlueLoading(2), Ready(3), GlueReady(4), Error(5), Invalid(6), Sealed(7), GlueSealed(8), Play(9)
Date - 2008-07-21
Time - 08:00:00.99
Logo, Title - 0/1/2 задает соответствующие флаги 0-None/1-Off/2-On
Command - строка команды, как в файле расписания
Например: "Shedule.ItemEx 0 5 3 2008-07-21 08:00:00.99 0 0 movie 0:00:02.00 [2.00] D:\TEMP\Glass.avi"

Приложение

Пример класса для приема и посылки сообщений (написан на Borland Builder)

```
class Sample {
public:

  TCOMISLMSConnection Connection;
  TCOMISLMSQueue Queue;

  SLMSErrorCode ErrorCode;
  HRESULT ResultCode

  // Соединиться с сервером, создать очередь для приема сообщений,
  // передать параметры для PostMessage по приходу сообщения
  bool Init(BSTR QueueName,
            HWND NotifyWindow, long NotifyMessage,
            long NotifyWParam,long NotifyLParam) {
    // создать сервер для связи
    ResultCode=CoSLMSConnection::Create(Connection);
    if (ResultCode!=S_OK) return 0;
    // создать очередь
    LPUNKNOWN ptr=0;
    HRESULT res=Connection->CreateQueue(&ErrorCode,&ResultCode,QueueName,
      (long)NotifyWindow,NotifyMessage,NotifyWParam,NotifyLParam,&ptr));
    if (res!=S_OK) {ResultCode=res; return 0;}
    if (ErrorCode!=mcSuccess) return 0;
    ResultCode=ptr->QueryInterface(IID_ISLMSQueue,(void**)&Queue));
    ptr->Release();
    if (ResultCode!=S_OK) return 0;
    // готов к работе с сообщениями
    return 1;
  }

  // Послать сообщение.
  // Успех означает, что сообщение попало к локальному серверу.
  // В случае ошибки при дальнейшем прохождении в очередь вернется
  //  сообщение об ошибке с ReplyTo==ThisMessId
  bool SendMessage(long ThisMessId, BSTR ToMachine, BSTR ToQueue,
                   BSTR Subject,BSTR Body) {
    HRESULT res=Queue->SendMessage(&ErrorCode,&ResultCode, ThisMessId,0,0,
                                   ToMachine,ToQueue, Subject, Body);
    if (res!=S_OK) {ResultCode=res; return 0;}
    if (ErrorCode!=mcSuccess) return 0;
    return 1;
  }

  // По получении определенного ранее оконного сообщения
  //  можно вычитать из очереди пришедшее сообщение
  void GetMessage() {
    SLMSMessageType MessType;
    long MessId,ReplyTo,SentLo,SentHi;
    BSTR FromMachine,FromQueue,Subject,Body;
    // получаем параметры сообщения
    HRESULT res=Queue->GetMessage(&ErrorCode,&ResultCode,
                                  &MessType,&MessId,&ReplyTo,
                                  &SentLo,&SentHi,
                                  &FromMachine,&FromQueue,
                                  &Subject,&Body);
    if (res!=S_OK || ErrorCode!=mcSuccess) {
      // Сообщение не получено, ничего освобождать не надо
      return;
    }
    // проверяем тип сообщения
    if (MessType==mtError) {
      // Сообщение об ошибке
      ВывестиКудато(ErrorCode,ResultCode,ReplyTo);
    } else if (MessType==mtMessage) {
      // Сообщение от кого-то
      ОбработатьСообщение(FromMachine,FromQueue,Subject,Body);
```

```
    } else {
      // Служебное сообщение
    }
    // обязательно освободить полученные от сервера строки
    SysFreeString(FromMachine);
    SysFreeString(FromQueue);
    SysFreeString(Subject);
    SysFreeString(Body);
  }

};
```

Пример программы для отправки сообщений (написан на MS Visual C++)

```
// отправка команды «запустить расписание» в экземпляр #1 программы FDOnAir

int _tmain(int argc, _TCHAR* argv[])
{
   // инициализировать COM
   ::CoInitialize(NULL);
   // создать главный объект
   CComPtr<ISLMSConnection> pConnection;
   if(S_OK != pConnection.CoCreateInstance(CLSID_SLMSConnection)) return -1;
   // создать очередь
   SLMSErrorCode ErrorCode;
   HRESULT ResultCode;
   HWND NotifyWindow = NULL; // окно не используется
   long NotifyCmd = WM_USER+100,NotifyWParam = 0,NotifyLParam = 0;
   CComBSTR bstrQName(_T("TestQueue")); // ВНИМАНИЕ: не "FDOnAir1" !
   CComPtr<IUnknown> pUnk;
   if(S_OK != pConnection->CreateQueue(&ErrorCode, &ResultCode, bstrQName,
      (long)NotifyWindow,NotifyCmd,NotifyWParam,NotifyLParam,&pUnk)) return -1;
   if(ErrorCode!=mcSuccess) return -1;
   // получить интерфейс ISLMSQueue
   CComQIPtr<ISLMSQueue> pQueue(pUnk);
   if(!pQueue) return -1;
   // отправить команду 'Запустить расписание' в FDOnAir #1
   CStringW queue_name = L"FDOnAir1";
   CStringW subject = L"OnAir1.Mirror";
   CStringW cmd = L"Shedule.Start";
   CStringW local = L"Localhost";
   // преобразовать строки в CComBSTR
   CComBSTR bToQueue(queue_name);
   CComBSTR bSubj(subject);
   CComBSTR bBody(cmd);
   CComBSTR bToMachine(local);
   long MessId = 1; // может быть использован любой ID
   pQueue->SLSendMessage(&ErrorCode,&ResultCode,MessId,0,0,bToMachine,bToQueue,bSubj,bBody);

   return 0;
}
```