**Модуль интеграции Infinity X  
Версия 1.0.40.Х**

В данном документе описаны способы решения основных задач по интеграции с «Infinity X» в соответствии с моделями First-Party и Third-Party. Используемые технологии: COM/ActiveX и .NET.

**Содержание документа**

* Модели интеграции
* Технологии интеграции
* Принцип интеграции
* Примеры решения типовых задач
  + Совершение исходящего вызова
* Детальное описание основных интерфейсов и функций
  + Интерфейсы COM/ActiveX
  + Интерфейсы .NET

**Модели интеграции**

Модель интеграции First-Party подразумевает управление системой от первого лица, т.е. осуществлять все действия от имени конкретного оператора. Как правило, данная модель используется при интеграции рабочих мест различных информационных систем с Infinity X.

Модель Third-Party позволяет управлять системой от третьего лица, т.е. осуществлять операции от имени всех операторов. Как правило, данная модель используется при интеграции сервера информационных систем с Infinity X.

Важно отметить, что модель First-Party предполагает, что рабочее место оператора не запущено. Через модуль интеграции осуществляется авторизация на сервере, установка статуса оператора, управление звонками и т.д. Фактически, информационная система полностью заменяет рабочее место оператора. Если же рабочее место оператора уже запущено и необходимо управлять им из информационной системы, необходимо использовать модель Third-Party.

Использование визуальных форм ActiveX возможно только в модели First-Party.

**Технологии интеграции**

Обе модели интеграции предполагают использование одной из двух технологий: COM/ActiveX или .NET.

Технология COM позволяет при помощи любого языка программирования обращаться к объектам системы и управлять ими. Технология ActiveX позволяет встраивать визуальные инструменты Infinity X в информационную систему. В частности, данные технологии удобно применять при интеграции Infinity X с платформой 1С, Lotus Notes, Excel и т.д.

Технология .NET подразумевает проведение интеграции с использованием средств разработки программного обеспечения, поддерживающих платформу .NET Framework. Как правило, это Microsoft Visual Studio (языки C#.NET, VB.NET, C++.NET), а также Embarcadero/Borland Delphi.

**Принцип интеграции**

Взаимодействие информационной системы с Infinity X состоит из следующих этапов:

1. Подготовительные операции
   1. Инициализация модуля интеграции
   2. Подключение к серверу Infinity X и авторизация
   3. Получение необходимых для решения конкретных задач интерфейсов
2. Основная содержательная часть
   1. Взаимодействие с интерфейсами (получение информации и осуществление операций)
3. Завершение работы
   1. Отключение от сервера Infinity X
   2. Освобождение ресурсов

Для использования технологии COM/ActiveX необходимо:

* Установить рабочее место Infinity X
* Зарегистрировать COM-объекты (запустить файл register.bat)
* Создать COM-объект "Cx.Integration.Core"

Для использования технологии .NET необходимо:

* Установить рабочее место Infinity X
* Создать проект на базе платформы .NET Framework 4.0
* Подключить к проекту следующие сборки:
  + Cx.Integration.AgatInfinityConnectorFactory.dll
  + Cx.Integration.AgatInfinityConnectorInterfaces.dll
* Создать экземпляр коннектора «Infinity X»

При необходимости подключения к серверу в режиме Third-Party необходимо создать пользователя, обладающего ролью «Интеграционное подключение Third-Party». Будем считать, что это пользователь «IntegrationUser» с паролем «1234», адрес сервера «192.168.200.101», порт 10010.

**Примеры решения типовых задач**

**1. Совершение исходящего вызова**

Задача: при запущенном рабочего месте оператора с внутренним номером «101» подключиться к серверу в режиме Third-Party и совершить от его имени исходящий вызов на номер «84956411010».

Пример использования технологии COM (содержимое файла MakeCall.vbs, который можно выполнить через командную строку Windows):

Set Srv = CreateObject("Cx.Integration.Core")

Srv.SetUseExceptions (True)

Dim LogonResult

LogonResult = Srv.LogonEx("IntegrationUser", "1234", "20000", "192.168.200.101", 10010)

If (Not Srv.IsConnected) Then

MsgBox Srv.LogonResultToString(LogonResult)

return

End If

Set CallManagement = Srv.GetCallManagement("101")

Call CallManagement.MakeCall("84956411010", "")

Пример использования технологии .NET (на языке C#):

IAgatInfinityConnectorFactory factory;

IAgatInfinityConnector connector;

private void button1\_Click(object sender, EventArgs e)

{

factory = AgatInfinityConnectorManager.GetAvailableFactories().FirstOrDefault(item => item.FactoryName == "Infinity X");

if (factory == null)

throw new Exception("Factory Infinity X not found");

connector = factory.CreateInstance(this);

connector.Connected += new EventHandler(connector\_Connected);

connector.Connect("Login:\"IntegrationUser\" Password:\"1234\" ServerAddress:\"192.168.200.101\" ServerPort:10010 BiDir", true);

}

void connector\_Connected(object sender, EventArgs e)

{

using (ICallManagement callManagement = connector.GetCallManagement("101"))

{

callManagement.MakeCall("84956411010", "");

}

connector.Disconnect();

connector = null;

factory = null;

}

**Детальное описание основных интерфейсов и функций**

**Интерфейсы COM/ActiveX**

Интерфейс ICxIntegrationCore

public interface ICxIntegrationCore

{

/// <summary>

/// Идентификатор экземпляра ядра (используется при отображении ActiveX-форм)

/// </summary>

[DispId(10000)]

string CoreID { get; }

/// <summary>

/// Установить режим обработки исключительных ситуаций

/// </summary>

/// <param name="Value">True - исключения включены, False - отключены</param>

[DispId(10001)]

void SetUseExceptions(bool Value);

/// <summary>

/// Описание последней ошибки

/// </summary>

[DispId(10002)]

string LastError { get; }

/// <summary>

/// Детальное описание последней ошибки

/// </summary>

[DispId(10003)]

string LastErrorDetailed { get; }

/// <summary>

/// Подключиться к серверу

/// </summary>

/// <param name="ConnectionString">Строка подключения</param>

[DispId(10004)]

void Connect(string ConnectionString);

/// <summary>

/// Подключиться к серверу, отобразив окно авторизации

/// </summary>

/// <returns></returns>

[DispId(10005)]

bool Logon();

/// <summary>

/// Подключиться к серверу и произвести авторизацию с использованием указанных параметров

/// </summary>

/// <param name="Login">Имя ползователя</param>

/// <param name="Password">Пароль</param>

/// <param name="Role">Роль "20000" - Администратор, "20004" - Оператор, "20010" - ThirdParty</param>

/// <param name="ServerAddress">Адрес сервера</param>

/// <param name="ServerPort">Порт сервера</param>

/// <returns>1 - успешно, остальное - код ошибки (см. LogonResult)</returns>

[DispId(10006)]

int LogonEx(string Login, string Password, string Role, string ServerAddress, int ServerPort);

/// <summary>

/// Выйти из системы

/// </summary>

[DispId(10007)]

void Logoff();

/// <summary>

/// Отключиться от сервера

/// </summary>

[DispId(10008)]

void Disconnect();

[DispId(10009)]

void Close();

/// <summary>

/// Возвращает состояние текущего подключения к серверу

/// </summary>

[DispId(10010)]

bool IsConnected { get; }

/// <summary>

/// Преобразовать код результата авторизации в строку

/// </summary>

/// <param name="Result">Результат авторизации</param>

/// <returns>Описание результата авторизации</returns>

[DispId(10011)]

string LogonResultToString(int Result);

/// <summary>

/// Получить интерфейс управления вызовами

/// </summary>

/// <param name="Extension">Внутренний номер абонента, соединениями которого необходимо

/// управлять (FirstParty). Если пусто - получаем интерфейс для управления всеми звонками

/// (ThirdParty)</param>

/// <returns>Интерфейс управления вызовами</returns>

[DispId(10012)]

IComCallManagement GetCallManagement(string Extension);

/// <summary>

/// Получить интерфейс доступа к информации о вызовах, соединениях и сеансах

/// </summary>

/// <returns>Интерфейс доступа к информации о вызовах, соединениях и сеансах</returns>

[DispId(10013)]

IComCallsConnectionsSeances GetCallsConnectionsSeances();

/// <summary>

/// Получить интерфейс управления пользователями

/// </summary>

/// <returns>Интерфейс управления пользователями</returns>

[DispId(10014)]

IComUsersManagement GetUsersManagement();

/// <summary>

/// Получить интерфейс управления кампаниями

/// </summary>

/// <returns>Интерфейс управления кампаниями</returns>

[DispId(10015)]

AgatInfinityConnector.ICampaignsManagement GetCampaignsManagement();

/// <summary>

/// Получить интерфейс утилит

/// </summary>

/// <returns>Интерфейс утилит</returns>

[DispId(10016)]

AgatInfinityConnector.IUtilsManager GetUtilsManager();

}

Интерфейс IComCallManagement

/// <summary>

/// Основной интерфейс для мониторинга вызовов (звонков).

/// </summary>

public interface IComCallManagement

{

/// <summary>

/// Внутренний номер - владелец вызовов. Может быть пустым, если получен интерфейс

/// для управления всеми вызовами.

/// </summary>

[DispId(10000)]

string Extension { get; }

/// <summary>

/// Все вызовы, включая находящиеся в состоянии "Завершен", "Занято", "Ошибка"

/// (хранятся около 30 секунд после завершения)

/// </summary>

[DispId(10001)]

object Calls { get; }

/// <summary>

/// Совершить новый вызов

/// </summary>

/// <param name="number\_">Номер телефона для набора</param>

/// <param name="callerName\_">Имя звонящего абонента (опционально)</param>

/// <param name="extension\_">Extension, от имени которого совершается вызов.

/// Нужно заполнять в случае, если интерфейс получен без указания Extension</param>

/// <returns>Интерфейс нового вызова</returns>

[DispId(10002)]

IComCall MakeCall(string number\_, string callerName\_, string extension\_ = "");

/// <summary>

/// Установить статус внутреннего номера

/// </summary>

/// <param name="state\_">Статус внутреннего номера</param>

/// <param name="extension\_">Extension, для которого нужно установить статус.

/// Нужно заполнять в случае, если интерфейс получен без указания Extension</param>

[DispId(10003)]

bool SetExtensionState(AgatInfinityConnector.ExtensionState state\_, string extension\_ = "");

/// Обработчики событий для скриптовых языков

[DispId(10004)]

object CallCreatedEvent { set; }

[DispId(10005)]

object CallDeletedEvent { set; }

[DispId(10006)]

object ExtensionStateChangedEvent { set; }

[DispId(10007)]

object DisposedEvent { set; }

[DispId(10008)]

object StateChangedEvent { set; }

[DispId(10009)]

object NumberChangedEvent { set; }

[DispId(10010)]

object NameChangedEvent { set; }

[DispId(10011)]

object DialedNumberChangedEvent { set; }

[DispId(10012)]

object CommandsStateChangedEvent { set; }

[DispId(10013)]

object DigitsSentEvent { set; }

[DispId(10014)]

object AbonentCallInfoChangedEvent { set; }

}

public interface ICallManagementEvents

{

/// <summary>

/// Создан новый вызов. На момент вызова события вызов уже содержится в коллекции Calls.

/// </summary>

[DispId(1001)]

void CallCreated(IComCall call\_);

/// <summary>

/// Вызов удален. На момент вызова события вызов еще содержится в коллекции Calls.

/// </summary>

[DispId(1002)]

void CallDeleted(IComCall call\_);

/// <summary>

/// Изменился статус внутреннего номера.

/// </summary>

[DispId(1003)]

void ExtensionStateChanged(string Extension, AgatInfinityConnector.ExtensionState State);

/// <summary>

/// Интерфейс уничтожен. Дальнейшая работа с ним невозможна.

/// </summary>

[DispId(1004)]

void Disposed();

/// <summary>

/// Состояние вызова изменилось

/// </summary>

[DispId(1005)]

void StateChanged(IComCall call\_, AgatInfinityConnector.CallState oldState, AgatInfinityConnector.CallState state);

/// <summary>

/// Номер абонента изменился

/// </summary>

[DispId(1006)]

void NumberChanged(IComCall call\_);

/// <summary>

/// Имя абонента изменилось

/// </summary>

[DispId(1007)]

void NameChanged(IComCall call\_);

/// <summary>

/// Набранный номер изменился

/// </summary>

[DispId(1008)]

void DialedNumberChanged(IComCall call\_);

/// <summary>

/// Состояние (доступность) команд изменилась

/// </summary>

[DispId(1009)]

void CommandsStateChanged(IComCall call\_);

/// <summary>

/// Выполнен тоновый донабор

/// </summary>

[DispId(1010)]

void DigitsSent(IComCall call\_, string digits);

/// <summary>

/// Состояние (доступность) команд изменилась

/// </summary>

[DispId(1011)]

void AbonentCallInfoChanged(IComCall call\_);

}

Интерфейс IComCall

public interface IComCall

{

/// <summary>

/// Уникальный идентификатор вызова. В зависимости от подключенного устройства,

/// уникальность может обеспечиваться абсолютно, либо в пределах времени работы устройства.

/// </summary>

[DispId(10000)]

string ID { get; }

/// <summary>

/// Уникальный идентификатор сеанса. Сеанс - это цепочка соединений, в которой участвует

/// данный вызов.

/// </summary>

[DispId(10001)]

string SeanceID { get; }

/// <summary>

/// Внутренний номер - владелец вызова

/// </summary>

[DispId(10002)]

string Extension { get; }

/// <summary>

/// Номер абонента

/// </summary>

[DispId(10003)]

string Number { get; }

/// <summary>

/// Имя абонента

/// </summary>

[DispId(10004)]

string Name { get; }

/// <summary>

/// Набранный номер (для входящих вызовов - номер городской телефонной линии)

/// </summary>

[DispId(10005)]

string DialedNumber { get; }

/// <summary>

/// Состояние вызова

/// </summary>

[DispId(10006)]

AgatInfinityConnector.CallState State { get; }

/// <summary>

/// Направление вызова

/// </summary>

[DispId(10007)]

AgatInfinityConnector.CallDirection Direction { get; }

/// <summary>

/// Время начала вызова

/// </summary>

[DispId(10008)]

DateTime StartTime { get; }

/// <summary>

/// Время последнего изменения состояния

/// </summary>

[DispId(10009)]

DateTime LastStateTime { get; }

/// <summary>

/// Продолжительность вызова (как правило, с момента последнего изменения состояния)

/// </summary>

[DispId(10010)]

TimeSpan Duration { get; }

/// <summary>

/// Информация о звонке

/// </summary>

[DispId(10011)]

string AbonentCallInfoStr { get; }

/// <summary>

/// Информация о звонке в виде COM-коллекции ( пары Key/Value )

/// В .Net лучше использовать ICall.AbonentCallInfo

/// </summary>

[DispId(10012)]

object AbonentCallInfoCollection { get; }

/// <summary>

/// Можно ли совершить новый вызов

/// </summary>

[DispId(10013)]

bool CanMake { get; }

/// <summary>

/// Можно ли завершить вызов

/// </summary>

[DispId(10014)]

bool CanDrop { get; }

/// <summary>

/// Можно ли ответить на вызов

/// </summary>

[DispId(10015)]

bool CanAnswer { get; }

/// <summary>

/// Можно ли поставить вызов на удержание

/// </summary>

[DispId(10016)]

bool CanHold { get; }

/// <summary>

/// Можно ли вернуть вызов с удержания

/// </summary>

[DispId(10017)]

bool CanUnHold { get; }

/// <summary>

/// Можно ли выполнить быстрый (слепой, безусловный) перевод вызова

/// </summary>

[DispId(10018)]

bool CanQuickTransfer { get; }

/// <summary>

/// Можно ли создать быструю (слепую, безусловную) конференцию

/// </summary>

[DispId(10019)]

bool CanQuickConference { get; }

/// <summary>

/// Можно ли выполнить обычный (с консультацией, условный) перевод вызова

/// </summary>

[DispId(10020)]

bool CanStartTransfer { get; }

/// <summary>

/// Можно ли завершить обычный перевод

/// </summary>

[DispId(10021)]

bool CanFinishTransfer { get; }

/// <summary>

/// Можно ли создать обычную (с консультацией, условную) конференцию

/// </summary>

[DispId(10022)]

bool CanStartConference { get; }

/// <summary>

/// Можно ли завершить создание конференции

/// </summary>

[DispId(10023)]

bool CanFinishConference { get; }

/// <summary>

/// Можно ли выполнить тоновый донабор цифр

/// </summary>

[DispId(10024)]

bool CanSendDigits { get; }

/// <summary>

/// Завершить вызов (положить трубку)

/// </summary>

[DispId(10025)]

bool Drop();

/// <summary>

/// Принять вызов (поднять трубку, ответить)

/// </summary>

[DispId(10026)]

bool Answer();

/// <summary>

/// Поставить вызов на удержание

/// </summary>

[DispId(10027)]

bool Hold();

/// <summary>

/// Вернуть вызов с удержания

/// </summary>

[DispId(10028)]

bool UnHold();

/// <summary>

/// Выполнить быстрый (слепой, безусловный) перевод вызова

/// </summary>

/// <param name="number\_">Номер телефона для набора</param>

/// <param name="callerName\_">Имя звонящего абонента (опционально)</param>

[DispId(10029)]

bool QuickTransfer(string number\_, string callerName\_ = "");

/// <summary>

/// Создать быструю (слепую, безусловную) конференцию

/// </summary>

/// <param name="number\_">Номер телефона для набора</param>

/// <param name="callerName\_">Имя звонящего абонента (опционально)</param>

[DispId(10030)]

bool QuickConference(string number\_, string callerName\_ = "");

/// <summary>

/// Выполнить обычный (с консультацией, условный) перевод вызова

/// </summary>

/// <param name="number\_">Номер телефона для набора</param>

/// <param name="callerName\_">Имя звонящего абонента (опционально)</param>

[DispId(10031)]

bool StartTransfer(string number\_, string callerName\_ = "");

/// <summary>

/// Завершить обычный перевод вызова

/// </summary>

[DispId(10032)]

bool FinishTransfer();

/// <summary>

/// Создать обычную (с консультацией, условную) конференцию

/// </summary>

/// <param name="number\_">Номер телефона для набора</param>

/// <param name="callerName\_">Имя звонящего абонента (опционально)</param>

[DispId(10033)]

bool StartConference(string number\_, string callerName\_ = "");

/// <summary>

/// Завершить создание конференции

/// </summary>

[DispId(10034)]

bool FinishConference();

/// <summary>

/// Выполнить тоновый донабор цифр

/// </summary>

[DispId(10035)]

bool SendDigits(string digits\_);

/// <summary>

/// Уникальный идентификатор родительского вызова. Используется для отслеживания цепочки если один вызов порождает другой (например, при переводе вызовов)

/// </summary>

[DispId(10036)]

string ParentCallID { get; }

}

Интерфейс IComUserManagement

/// <summary>

/// Основной интерфейс для мониторинга пользователей.

/// </summary>

public interface IComUsersManagement

{

/// <summary>

/// Коллекция пользователей. В коллекции присутствуют все пользователи независимо от их

/// текущих статусов.

/// </summary>

[DispId(10000)]

object Users { get; }

/// <summary>

/// Текущий залогиненный пользователь.

/// </summary>

[DispId(10001)]

IComUser CurrentUser { get; }

/// <summary>

/// Обработчики событий для скриптовых языков

/// </summary>

[DispId(10002)]

object StateChangedEvent { set; }

}

public interface IUsersManagementEvents

{

/// <summary>

/// Событие возникает при изменении статуса пользователя

/// </summary>

[DispId(1001)]

void StateChanged(IComUser User, object OldState, object State);

}

Интерфейс IComUser

/// <summary>

/// Основной интерфейс пользователя, предназначенный для получения информации и управления

/// </summary>

public interface IComUser

{

/// <summary>

/// Идентификатор пользователя

/// </summary>

[DispId(10000)]

long ID { get; }

/// <summary>

/// Имя пользователя, используемое для входа в систему

/// </summary>

[DispId(10001)]

string Login { get; }

/// <summary>

/// Текущий статус пользователя

/// </summary>

[DispId(10002)]

long State { get; set; }

//UserState State { get; set; }

/// <summary>

/// Список внутренних телефонных номеров пользователя

/// </summary>

[DispId(10003)]

object Extensions { get; }

/// <summary>

/// Вошел ли пользователь в систему

/// </summary>

[DispId(10004)]

bool IsLoggedIn { get; }

/// <summary>

/// Войти в систему. Возвращает результат (успешно/неуспешно)

/// </summary>

[DispId(10005)]

bool Logon(string Password, object IDRole);

[DispId(10006)]

long LogonEx(string Password, object IDRole);

/// <summary>

/// Выйти из системы

/// </summary>

[DispId(10007)]

void Logoff();

[DispId(10008)]

void SetStateOnline();

[DispId(10009)]

void SetStateBreak();

[DispId(10010)]

void SetStateAway();

[DispId(10011)]

void SetStateNotAvailable();

}

Интерфейс ICampaignsManagement

/// <summary>

/// Основной интерфейс для работы с кампаниями

/// </summary>

[ComVisible(true)]

public interface ICampaignsManagement : IDisposable

{

/// <summary>

/// Запустить кампанию

/// </summary>

void StartCampaign(long IDCampaign);

/// <summary>

/// Остановить кампанию

/// </summary>

void StopCampaign(long IDCampaign, bool bForce);

/// <summary>

/// Получить статус кампании

/// </summary>

uint GetCampaignStatus(long IDCampaign);

}

Интерфейс IComCallsConnectionsSeances

/// Основной интерфейс для доступа к информации о вызовах, соединениях и сеансах

public interface IComCallsConnectionsSeances

{

/// <summary>

/// Получить набор соединений по идентификатору сеанса

/// </summary>

/// <param name="seanceID\_">Идентификатор сеанса</param>

/// <returns>Набор соединений</returns>

[DispId(10000)]

object GetConnectionsBySeanceID(string seanceID\_);

/// <summary>

/// Получить набор соединений по идентификатору Call-а

/// </summary>

/// <param name="seanceID\_">Идентификатор Call-а</param>

/// <returns>Набор соединений</returns>

[DispId(10001)]

object GetConnectionsByCallID(string CallID);

/// <summary>

/// Получить набор соединений по диапазону времени, номеру телефона, идентификатору линии

/// </summary>

/// <param name="timeStartFrom\_">Начало диапазона времени</param>

/// <param name="timeStartTo\_">Окончание диапазона времени</param>

/// <param name="number\_">Фрагмент номера телефона</param>

/// <param name="lineID\_">Идентификатор линии</param>

/// <returns>Набор соединений</returns>

[DispId(10002)]

object GetConnections(DateTime timeStartFrom\_, DateTime timeStartTo\_, string number\_, string lineID\_);

/// <summary>

/// Получить соединение по идентификатору

/// </summary>

/// <param name="connectionID\_">Идентификатор соединения</param>

/// <returns>Соединение</returns>

[DispId(10003)]

object GetConnectionByID(long connectionID\_);

/// <summary>

/// Получить набор звонков по диапазону времени и пользователю

/// </summary>

/// <param name="timeStartFrom\_">Начало диапазона времени</param>

/// <param name="timeStartTo\_">Окончание диапазона времени</param>

/// <param name="IDUser\_">Пользователь</param>

/// <returns></returns>

[DispId(10004)]

object GetCalls(DateTime timeStartFrom\_, DateTime timeStartTo\_, long IDUser\_);

/// <summary>

/// Получить текущий набор соединений

/// </summary>

[DispId(10005)]

object GetCurrentConnections();

/// Обработчики событий для скриптовых языков

[DispId(20001)]

object ConnectionCreatedEvent { set; }

[DispId(20002)]

object ConnectionDeletedEvent { set; }

[DispId(20003)]

object ConnectionChangedEvent { set; }

}

// IComCallsConnectionsSeances Events

public interface IComCallsConnectionsSeancesEvents

{

/// <summary>

/// Создано новое соединение.

/// </summary>

[DispId(1001)]

void ConnectionCreated(AgatInfinityConnector.IConnection Connection);

/// <summary>

/// Соединение удалено.

/// </summary>

[DispId(1002)]

void ConnectionDeleted(AgatInfinityConnector.IConnection Connection);

/// <summary>

/// Соединение изменено.

/// </summary>

[DispId(1003)]

void ConnectionChanged(AgatInfinityConnector.IConnection Connection);

}

Интерфейс IConnection

public interface IConnection

{

/// <summary>

/// Уникальный идентификатор соединения

/// </summary>

[DispId(10000)]

long ID { get; }

/// <summary>

/// Время начала соединения

/// </summary>

[DispId(10001)]

DateTime TimeStart { get; }

/// <summary>

/// Продолжительность разговора

/// </summary>

[DispId(10002)]

TimeSpan DurationTalk { get; }

/// <summary>

/// Состояние соединения

/// </summary>

[DispId(10003)]

ConnectionState State { get; }

/// <summary>

/// Номер А

/// </summary>

[DispId(10004)]

string ANumber { get; }

/// <summary>

/// Номер Б

/// </summary>

[DispId(10005)]

string BNumber { get; }

/// <summary>

/// Абонент А

/// </summary>

[DispId(10006)]

string ADisplayText { get; }

/// <summary>

/// Абонент Б

/// </summary>

[DispId(10007)]

string BDisplayText { get; }

/// <summary>

/// Признак наличия записанного разговора

/// </summary>

[DispId(10008)]

bool IsRecorded { get; }

/// <summary>

/// Уникальный идентификатор соединения

/// </summary>

[DispId(10014)]

object ID\_AsVariant { get; }

/// <summary>

/// Сохранить записанный разговор в предоставленный поток

/// </summary>

/// <param name="stream\_">Поток для записи разговора</param>

[DispId(10009)]

void SaveRecordedFileToStream(Stream stream\_);

/// <summary>

/// Сохранить записанный разговор в указанный файл

/// </summary>

/// <param name="fileName\_">Имя файла</param>

[DispId(10010)]

void SaveRecordedFile(string fileName\_);

/// <summary>

/// Воспроизвести записанный разговор через ShellExecute, сохранив его во временной папке

/// </summary>

[DispId(10011)]

void PlayRecordedFile();

/// <summary>

/// Начать запись разговора

/// </summary>

[DispId(10012)]

void StartRecord();

/// <summary>

/// Остановить запись разговора

/// </summary>

[DispId(10013)]

void StopRecord();

}

Интерфейс ICampaignsManagement

public interface ICampaignsManagement

{

/// <summary>

/// Запустить кампанию

/// </summary>

[DispId(10000)]

void StartCampaign(long IDCampaign);

/// <summary>

/// Остановить кампанию

/// </summary>

[DispId(10001)]

void StopCampaign(long IDCampaign, bool bForce);

/// <summary>

/// Получить статус кампании

/// </summary>

[DispId(10002)]

uint GetCampaignStatus(long IDCampaign);

}

Интерфейс IUtilsManager

public interface IUtilsManager

{

// Configuration parameters

[DispId(10000)]

object GetConfigurationParameter(string Name, long IDObject);

[DispId(10001)]

bool TryGetConfigurationParameter(string Name, long IDObject, out object Value);

[DispId(10002)]

void SetConfigurationParameter(string Name, long IDObject, object Value);

[ComVisible(false)]

List<object> GetConfigurationParameters(List<string> Names, long IDObject);

[ComVisible(false)]

void SetConfigurationParameters(List<string> Names, long IDObject, List<object> Values);

}

Вспомогательные типы данных

enum LogonResult

{

// Логон не выполнялся

Uninitialized = 0,

// Логон завершен успешно

OK = 1,

// Ошибка на сервере, не понятно какая...

Error = 2,

// Неверный логин или пароль. Для logoff - пользователь не найден.

Invalid = 3,

// Уже залогинен.

AlreadyLogined = 4,

// Доступ запрещен с такими правами.

AccessDenied = 5,

// Возвращен список ролей. Нужно выбрать роль.

RolesList = 6,

// Нет лицензии (для переданного RoleType)

LicenseDenied = 7,

// Несоответсивие версий клиента и сервера

WrongVersion = 10,

// Ограниченный режим лицензии (запуск с 2-мя инструментами)

LicenceRestricted = 13

};

public enum CallState

{

/// <summary>

/// Неизвестно

/// </summary>

Unknown = 0,

/// <summary>

/// Входящий звонок (телефонный аппарат сейчас звенит)

/// </summary>

Ringing = 1,

/// <summary>

/// Напоминание (входящий звонок на второй линии, возврат звонка и т.д.)

/// </summary>

Reminder = 2,

/// <summary>

/// Гудок (трубка снята, ожидание набора номера). В Infinity 4 не используется

/// </summary>

Dialtone = 11,

/// <summary>

/// Набор номера

/// </summary>

DialNumber = 12,

/// <summary>

/// Ожидание ответа (аналог сигнала КПВ). В Infinity 4 не используется

/// </summary>

Alerting = 13,

/// <summary>

/// На линии (идет разговор)

/// </summary>

Connected = 21,

/// <summary>

/// Конференция, активная сторона (можно управлять этой конференцией)

/// </summary>

Conference = 31,

/// <summary>

/// В конференции, пассивная сторона (нельзя управлять этой конференцией, кроме отбоя). В Infinity 4 не используется (вместо InConference будет Connected)

/// </summary>

InConference = 32,

/// <summary>

/// Удержание, активная сторона (можно управлять этим удержанием)

/// </summary>

Hold = 41,

/// <summary>

/// На удержании, пассивная сторона (нельзя управлять этим удержанием, кроме отбоя)

/// </summary>

OnHold = 42,

/// <summary>

/// Перевод. В Infinity 4 не используется (вместо InConference используется Hold)

/// </summary>

Transfer = 51,

/// <summary>

/// Завершен

/// </summary>

Finished = 100

}

/// <summary>

/// Направление вызова

/// </summary>

[System.Runtime.InteropServices.ComVisible(true)]

[System.Runtime.InteropServices.Guid("C7FB225C-2CD6-4B11-926D-670EC597131C")]

public enum CallDirection

{

/// <summary>

/// Неизвестно

/// </summary>

Unknown = 0,

/// <summary>

/// Входящий

/// </summary>

In = 1,

/// <summary>

/// Исходящий

/// </summary>

Out = 2

}

/// <summary>

/// Статус пользователя

/// </summary>

public enum UserState: long

{

/// <summary>

/// Неизвестно

/// </summary>

Unknown = 0,

/// <summary>

/// Отключен (не в сети)

/// </summary>

Offline = 399,

/// <summary>

/// На месте

/// </summary>

Online = 300,

/// <summary>

/// Перерыв (поствызывная обработка)

/// </summary>

Timeout = 301,

/// <summary>

/// Отошел

/// </summary>

Away = 302,

/// <summary>

/// Недоступен

/// </summary>

NotAvailable = 303

}

**Интерфейсы .NET**

Для части интерфейсов базовые интерфейсы описаны в разделе «**Интерфейсы COM/ActiveX**»

Класс AgatInfinityConnectorManager

/// <summary>

/// Менеджер коннекторов - основной класс, с помощью которого можно получить

/// список доступных фабрик коннекторов

/// </summary>

public static class AgatInfinityConnectorManager

{

/// <summary>

/// Возвращает список доступных фабрик коннекторов. Каждая фабрика - это плагин,

/// расположенный рядом с этой сборкой. В ближайшее время предполагается три фабрики:

/// Infinity 4

/// Infinity X

/// IP-АТС Agat-UX

/// </summary>

/// <returns>Список фабрик</returns>

public static IEnumerable<IAgatInfinityConnectorFactory> GetAvailableFactories(

string pluginsPath\_ = "",

string logFileName\_ = "");

}

Интерфейс IAgatInfinityConnectorFactory

/// <summary>

/// Основной интерфейс фабрики, предназначенный для создания экземпляра коннектора

/// </summary>

public interface IAgatInfinityConnectorFactory

{

/// <summary>

/// Создает экземпляр коннектора

/// </summary>

/// <param name="callbackControl\_">Элемент управления, в контексте потока которого будут

/// приходить все события</param>

/// <returns>Интерфейс коннектора</returns>

IAgatInfinityConnector CreateInstance(System.Windows.Forms.Control callbackControl\_);

/// <summary>

/// Возвращает имя фабрики

/// </summary>

string FactoryName { get; }

}

Интерфейс IAgatInfinityConnector

/// <summary>

/// Основной интерфейс коннектора, предназначенный для управления соединением с сервером/устройством

/// </summary>

public interface IAgatInfinityConnector : IDisposable

{

/// <summary>

/// Установить подключение к серверу/устройству

/// </summary>

/// <param name="connectionString\_">Строка подключения содержит адрес сервера, порт, логин, пароль

/// и другую информацию, которая необходима для подключения к системе</param>

/// <param name="alwaysKeepConnection\_">Автоматически переподключаться при обрыве

/// соединения</param>

void Connect(string connectionString\_, bool alwaysKeepConnection\_);

/// <summary>

/// Отключиться от сервера/устройства

/// </summary>

void Disconnect();

/// <summary>

/// Текущее состояние подключения - подключен/отключен

/// </summary>

bool IsConnected { get; }

/// <summary>

/// Получить список доступных внутренних номеров (Extension)

/// </summary>

IEnumerable<string> GetExtensions();

/// <summary>

/// Получить интерфейс управления вызовами

/// </summary>

/// <param name="extension\_">Внутренний номер абонента, соединениями которого необходимо

/// управлять (FirstParty). Если пусто - получаем интерфейс для управления всеми звонками

/// (ThirdParty)</param>

/// <param name="password\_">Пароль (используется только в режиме FirstParty)</param>

/// <returns>Интерфейс управления вызовами</returns>

ICallManagement GetCallManagement(string extension\_, string password\_ = "");

/// <summary>

/// Получить интерфейс управления пользователями

/// </summary>

/// <returns>Интерфейс управления пользователями</returns>

IUserManagement GetUserManagement();

/// <summary>

/// Получить интерфейс управления кампаниями

/// </summary>

/// <returns>Интерфейс управления кампаниями</returns>

ICampaignsManagement GetCampaignsManagement();

/// <summary>

/// Получить строку подключения

/// </summary>

string ConnectionString { get; }

/// <summary>

/// Соединение с сервером (станцией) установлено

/// </summary>

event EventHandler Connected;

/// <summary>

/// Соединение с сервером (станцией) прервано

/// </summary>

event EventHandler Disconnected;

}

Интерфейс ICallManagement

/// <summary>

/// Основной интерфейс для мониторинга вызовов (звонков). Мониторинг прекращается при вызове Dispose()

/// </summary>

public interface ICallManagement : IDisposable

{

/// <summary>

/// Внутренний номер - владелец вызовов. Может быть пустым, если получен интерфейс для управления

/// всеми вызовами.

/// </summary>

string Extension { get; }

/// <summary>

/// Все вызовы, включая находящиеся в состоянии "Завершен", "Занято", "Ошибка" (хранятся около

/// 30 секунд после завершения)

/// </summary>

IEnumerable<ICall> Calls { get; }

/// <summary>

/// Совершить новый вызов

/// </summary>

/// <param name="number\_">Номер телефона для набора</param>

/// <param name="callerName\_">Имя звонящего абонента (опционально)</param>

/// <param name="extension\_">Extension, от имени которого совершается вызов. Нужно заполнять

/// в случае, если интерфейс получен без указания Extension</param>

/// <returns>Интерфейс нового вызова</returns>

ICall MakeCall(string number\_, string callerName\_, string extension\_ = "");

/// <summary>

/// Создан новый вызов. На момент вызова события вызов уже содержится в коллекции Calls.

/// </summary>

event EventHandler<CallEventArgs> CallCreated;

/// <summary>

/// Вызов удален. На момент вызова события вызов еще содержится в коллекции Calls.

/// </summary>

event EventHandler<CallEventArgs> CallDeleted;

/// <summary>

/// Интерфейс уничтожен. Дальнейшая работа с ним невозможна.

/// </summary>

event EventHandler Disposed;

}

Интерфейс ICall

/// <summary>

/// Основной интерфейс вызова (звонка), предназначенный для получения информации и управления

/// </summary>

public interface ICall: Cx.ActiveX.IComCall

{

/// <summary>

/// Основной интерфейс управления вызовами, который породил данный вызов

/// </summary>

ICallManagement CallManagement { get; }

/// <summary>

/// Информация о звонке

/// </summary>

List<KeyValuePair<string, string>> AbonentCallInfo { get; }

/// <summary>

/// Состояние вызова изменилось

/// </summary>

event EventHandler<CallStateEventArgs> StateChanged;

/// <summary>

/// Номер абонента изменился

/// </summary>

event EventHandler<CallEventArgs> NumberChanged;

/// <summary>

/// Имя абонента изменилось

/// </summary>

event EventHandler<CallEventArgs> NameChanged;

/// <summary>

/// Набранный номер изменился

/// </summary>

event EventHandler<CallEventArgs> DialedNumberChanged;

/// <summary>

/// Состояние (доступность) команд изменилась

/// </summary>

event EventHandler<CallEventArgs> CommandsStateChanged;

/// <summary>

/// Выполнен тоновый донабор

/// </summary>

event EventHandler<CallDigitsSentEventArgs> DigitsSent;

/// <summary>

/// Информация о звонке изменилась

/// </summary>

event EventHandler<CallEventArgs> AbonentCallInfoChanged;

}

Интерфейс IUserManagement

/// <summary>

/// Основной интерфейс для мониторинга пользователей. Мониторинг прекращается при вызове Dispose()

/// </summary>

public interface IUserManagement : IDisposable

{

/// <summary>

/// Коллекция пользователей. В коллекции присутствуют все пользователи независимо от их

/// текущих статусов.

/// </summary>

IEnumerable<IUser> Users { get; }

/// <summary>

/// Текущий залогиненный пользователь.

/// </summary>

IUser CurrentUser { get; }

/// <summary>

/// Событие возникает при изменении статуса пользователя

/// </summary>

event EventHandler<UserStateEventArgs> StateChanged;

}

Интерфейс IUser

/// <summary>

/// Основной интерфейс пользователя, предназначенный для получения информации и управления

/// </summary>

public interface IUser : Cx.ActiveX.IComUser

{

/// <summary>

/// Список внутренних телефонных номеров пользователя

/// </summary>

new IEnumerable<string> Extensions { get; }

/// <summary>

/// Событие возникает при изменении статуса пользователя

/// </summary>

event EventHandler<UserStateEventArgs> StateChanged;

}

Интерфейс ICampaignsManagement

/// <summary>

/// Основной интерфейс для работы с кампаниями

/// </summary>

[ComVisible(true)]

public interface ICampaignsManagement : IDisposable

{

/// <summary>

/// Запустить кампанию

/// </summary>

void StartCampaign(long IDCampaign);

/// <summary>

/// Остановить кампанию

/// </summary>

void StopCampaign(long IDCampaign, bool bForce);

/// <summary>

/// Получить статус кампании

/// </summary>

uint GetCampaignStatus(long IDCampaign);

}

Интерфейс IUtilsManager

public interface IUtilsManager

{

// Configuration parameters

[DispId(10000)]

object GetConfigurationParameter(string Name, long IDObject);

[DispId(10001)]

bool TryGetConfigurationParameter(string Name, long IDObject, out object Value);

[DispId(10002)]

void SetConfigurationParameter(string Name, long IDObject, object Value);

[ComVisible(false)]

List<object> GetConfigurationParameters(List<string> Names, long IDObject);

[ComVisible(false)]

void SetConfigurationParameters(List<string> Names, long IDObject, List<object> Values);

}

Вспомогательные типы данных

/// <summary>

/// Информация о событии создания/удаления вызова

/// </summary>

public class CallEventArgs : EventArgs

{

/// <summary>

/// Вызов, к которому относится событие

/// </summary>

public ICall Call { get; private set; }

}

/// <summary>

/// Информация о событии изменения статуса внутреннего номера

/// </summary>

public class ExtensionStateChangedArgs : EventArgs

{

/// <summary>

/// Внутренний номер

/// </summary>

public string Extension { get; private set; }

/// <summary>

/// Статус

/// </summary>

public ExtensionState State { get; private set; }

}

/// <summary>

/// Информация о событии изменения состояния вызова

/// </summary>

public class CallStateEventArgs : EventArgs

{

/// <summary>

/// Вызов, породивший событие

/// </summary>

public ICall Call { get; private set; }

/// <summary>

/// Предыдущее состояние вызова

/// </summary>

public CallState OldState { get; private set; }

/// <summary>

/// Новое состояние вызова

/// </summary>

public CallState NewState { get; private set; }

}

/// <summary>

/// Информация о событии о измении информации о пользователе

/// </summary>

public class UserEventArgs : EventArgs

{

/// <summary>

/// Основной конструктор

/// </summary>

/// <param name="user\_">Пользователь, породивший событие</param>

public UserEventArgs(IUser user\_)

{

User = user\_;

}

/// <summary>

/// Пользователь, породивший событие

/// </summary>

public IUser User { get; private set; }

}

/// <summary>

/// Информация о событии изменения статуса пользователя

/// </summary>

public class UserStateEventArgs : UserEventArgs

{

/// <summary>

/// Основной конструктор

/// </summary>

/// <param name="user\_">Пользователь, породивший событие</param>

/// <param name="oldState\_">Предыдущий статус пользователя</param>

/// <param name="newState\_">Новый статус пользователя</param>

public UserStateEventArgs(IUser user\_, long oldState\_, long newState\_):

base(user\_)

{

OldState = oldState\_;

NewState = newState\_;

}

/// <summary>

/// Предыдущий статус пользователя

/// </summary>

public long OldState { get; private set; }

/// <summary>

/// Новый статус пользователя

/// </summary>

public long NewState { get; private set; }

}