Developer Guide

PhoneX

AVAYA

Contact Center Express

Release 2.0 - Issue 0

Notice

Every effort was made to ensure that the information in this manual was complete and accurate at the time of printing. However, information is subject to change.

Your Responsibility for Your System's Security

Toll fraud is the unauthorized use of your telecommunications system by an unauthorized party, for example, persons other than your company's employees, agents, subcontractors, or persons working on your company's behalf. Note that there may be a risk of toll fraud associated with your telecommunications system and, if toll fraud occurs, it can result in substantial additional charges for your telecommunications services.

You and your system manager are responsible for the security of your system, such as programming and configuring your equipment to prevent unauthorized use. The system manager is also responsible for reading all installation, instruction, and system administration documents provided with this product in order to fully understand the features that can introduce risk of toll fraud and the steps that can be taken to reduce that risk. Avaya Inc. does not warrant that this product is immune from or will prevent unauthorized use of common-carrier telecommunication services or facilities accessed through or connected to it. Avaya Inc. will not be responsible for any charges that result from such unauthorized use.

Avaya Fraud Intervention

If you suspect that you are being victimized by toll fraud and you need technical support or assistance, call the Technical Service Center Toll Fraud Intervention Hot-line at 1-800-643-2353.

Trademarks

- Avaya Computer Telephony is a registered trademark of Avaya Inc.
- Avaya CallMaster is a registered trademark of Avaya Inc.
- Definity is a registered trademark of Avaya Inc.
- MultiVantage is a registered trademark of Avaya Inc.
- INTEL and Pentium are registered trademarks of Intel Corporation.
- Microsoft, MS, MS-DOS, and Windows are registered trademarks of Microsoft Corp.

All other product names mentioned herein are the trademarks of their respective owners.

Avaya National Customer Care Center

Avaya provides a telephone number for you to use to report problems or to ask questions about your contact center. The support telephone number is 1-800-242-2121. For technical support, customers outside the United States should call their Avaya representative or distributor.

European Union Declaration of Conformity

Avaya Inc. Business Communications Systems declares that the equipment specified in this document conforms to the referenced European Union (EU) Directives and Harmonized Standards listed below:

▪ EMC Directive	89/336/EEC
▪ Low Voltage Directive	73/23/EEC

The CE” mark affixed to the equipment means that it conforms to the above Directives.

Website

For more information on all Avaya Contact Center Express products, refer to the company ***website*** (http://www.AvayaContactCenterExpress.com).

Software License Agreement

Definitions

Term	Definition
Avaya	Avaya Inc.
You, your or licensee	The person or business entity who purchased this license to use this client software or for whom such license was purchased.
Client software	A software application that operates on a computer system.
Documentation	The manual and any other printed material provided by Avaya for the client software.
License	The license purchased and granted pursuant to this agreement.

License and Protection

License Grant. Avaya grants to you, subject to the following terms and conditions, a nonexclusive, nontransferable right to use the client software on one or more single-user devices. The maximum simultaneous users of the client software being limited to the number of single-user licenses purchased and owned by you. Avaya reserves all rights not expressly granted to you.

Protection of Software. You agree to take all reasonable steps to protect the client software and documentation from unauthorized copy or use. The client software source code represents and embodies trade secrets of Avaya and/or its licensors. The source code and embodied trade secrets are not licensed to you and any modification, addition, or deletion is strictly prohibited. You agree not to disassemble, decompile, or otherwise reverse engineer the client software in order to discover the source code and/or the trade secrets contained in the source code or for any other reason. To the extent that the client software is located in a Member State of the European Community and you need information about the client software in order to achieve interoperability of an independently created software program with the client software, you shall first request such information from Avaya. Unless Avaya refuses to make such information available, you shall not take any steps, such as reverse assembly or reverse compilation, to derive a source code equivalent to the client software. Avaya may charge you a reasonable fee for the provision of such information.

Copies. You may make multiple copies of the client software for your own use with Avaya contact center agent digital voice terminals, provided you do not violate the License Grant in paragraph 1, and you do not receive any payment, commercial benefit, or other consideration for reproduction or use. You may not copy documentation unless it carries a statement that copying is permitted. All proprietary rights notices must be faithfully reproduced and included on all copies.

Ownership. Ownership of, and title to, the client software and documentation (including any adaptations or copies) remains with Avaya and/or its licensors.

Restrictions. You agree not to rent, lease, sublicense, modify or time share the client software or documentation.

Termination. This agreement shall automatically terminate if you breach any of the terms or conditions of this agreement. You agree to destroy the original and all copies of the client software and documentation, or to return them to Avaya, upon termination of this license.

Limited Warranty and Limited Liability

Compatibility. The client software is only compatible with certain computers and operating systems. The software is not warranted for noncompatible systems.

Software. Avaya warrants that if the client software fails to substantially conform to the specifications in the documentation and if the client software is returned to the place from which it was purchased within one (1) year from the date purchased, then Avaya will either replace the client software or offer to refund the license fee to you upon return of all copies of the client software and documentation to Avaya. In the event of a refund, the license shall terminate.

Disclaimer of Warranties. Avaya makes no warranty, representation or promise not expressly set forth in this agreement. Avaya disclaims and excludes any and all implied warranties of merchantability or fitness for a particular purpose. Avaya does not warrant that the client software or documentation will satisfy your requirements or that the client software or documentation are without defect or error or that the operation of the software will be uninterrupted. Some states or countries do not allow the exclusion of implied warranties or limitations on how long an implied warranty lasts, so the above limitation may not apply to you. This warranty gives you specific legal rights which vary from state to state.

Exclusive Remedy. Except for bodily injury caused by Avaya's negligence, Avaya's entire liability arising from or relating to this agreement or the client software or documentation and your exclusive remedy is limited to direct damages in an amount not to exceed $10,000. Avaya shall not in any case be liable for any special incidental, consequential, indirect or punitive damages even if Avaya has been advised of the possibility of such damages. Avaya is not responsible for lost profits or revenue, loss of use of the client software, loss of data, costs of recreating lost data, the cost of any substitute equipment or program, or claims by any party other than you. Some states or countries do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

General Conditions

Governing Law. This agreement shall be governed by, and interpreted in accordance with, the substantive laws of the State of New Jersey of the United States of America.

Entire Agreement. This agreement sets forth the entire understanding and agreement between you and Avaya and may be amended only in a writing or writings signed by you and Avaya. No vendor, distributor, dealer, retailer, sales person or other person is authorized to modify this agreement or to make any warranty, representation or promise which is different than, or in addition to, the representations or promises of this agreement about the software.

Export. Licensee hereby agrees that it will not knowingly, directly or indirectly, without prior written consent, if required, of the Office of Export Licensing of the U.S. Department of Commerce, Washington D.C. 20230, export or transmit any of the Products to any group Q, S, W, Y, or Z country specified in the Export Administration Regulations issued by the U.S. Department of Commerce or to any country which such transmission is restricted by applicable regulations or statues.

U.S. Government Restricted Rights. Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in FAR 52.227-14 (June 1987) Alternate III (g)(3) (June 1987), FAR 52.227-19 (June 1987), or DFARS 52.227-7013 (c)(1)(ii) (June 1988), as applicable Contractor/Manufacturer is Avaya Inc. 11900 North Pecos Street, Westminster, Colorado 80234.

Assignment. Avaya may without your consent or notice to you, assign this agreement to an entity to which it transfers ownership of the client software. Upon the effective date of such assignment, you agree that Avaya shall be released and discharged from all obligations and liabilities under this agreement.

Contents

CHAPTER 1

Preface

This chapter provides information that will help you use this document.

In This Chapter

Document Conventions

Convention	Description
CallDial	Italicized text indicates a reference to a method, event, control property parameter, class variable, return value or enumeration.
`Sample code`	Courier font indicates sample program code.

Related Documents

This document should be used in conjunction with the *Definity ECS Programmer’s Guide for CentreVu CTI*, which provides more extensive information on the API calls used.

The *Definity Enterprise Communications Server Release 9 Administrator’s Guide, Vols 1,2 & 3 (555-233-506)* is helpful when administering the Definity Enterprise Communications Server.

For this document and more documentation on the Definity Enterprise Communications Server, refer to ***Avaya's Support Center website*** (see Product Documentation - http://support.avaya.com.).

Knowledge Base

For information on any errors and updates relating to this document, visit the Avaya Contact Center Express Knowledge Base via the ***website*** (http://www.AvayaContactCenterExpress.com).

On-Line Help

To display on-line help on all the properties, methods and events exposed by this control, select the control after you have placed it on a form and press the [F1] key.

CHAPTER 2

Introduction

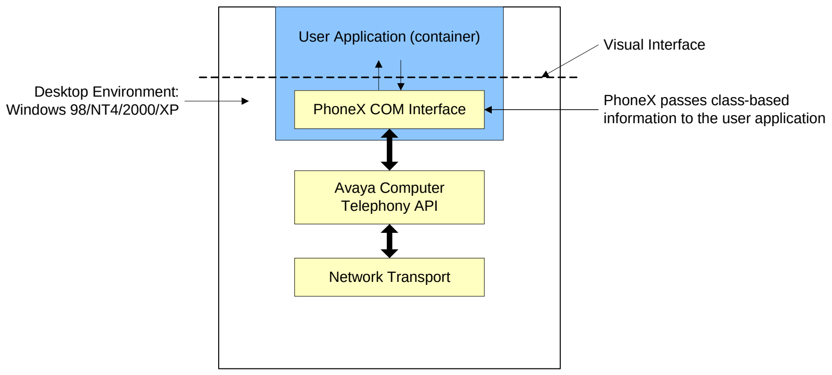
This chapter introduces PhoneX and illustrates how it interfaces with a user application.

In This Chapter

What is PhoneX?

PhoneX is an OCX control that performs general telephony control. It exists as a wrapper control around the Avaya Computer Telephony middleware, offering an abstract level of call, device, agent and scripting control for common desktop applications and development environments.

PhoneX presents an extensive array of methods, events and properties to a container application. It has no user interface and relies entirely on the host application to provide this.


User Application (container)
Visual Interface
Desktop Environment: Windows 98/NT4/2000/XP
PhoneX COM Interface
PhoneX passes class-based information to the user application
Avaya Computer Telephony API
Network Transport

Sample Code

Sample Visual Basic code is included in this document to illustrate the use of the PhoneX API interface. The code should be easily transported to applications written in Microsoft Visual Basic (version 5 or greater), VBA, HTML (using VBScript) and others.

CHAPTER 3

The Class Structures

This chapter includes information on the class structures used within PhoneX.

In This Chapter

Introduction

General parameter passing is based around class objects with the addition of specific parameters as the method or event requires.

The object model allows PhoneX to hold copies of all class objects used in call, device and agent methods.

There are four primary classes (Call, Device, Agent and Error) and five collection classes (*ActiveCallClasses*, *OldCallClasses*, *AgentClasses*, *DevicesClasses* and *TServers*).

CallClass

The CallClass contains all information about an individual call as seen from the perspective of an individual device. This means that if PhoneX is used to monitor two devices, Party A and Party B, and Party A and Party B are involved on the same call, PhoneX will hold two call classes. One class views the call from Party A's perspective with Party A as the primary device number, and one class views the call from Party B's perspective with Party B as the primary device number.

CallClass Parameters

Class Parameter	Type	Default Value
BillRate		
BillType	*Long*	
CallAppearance	*Long, Read/write*	*Default: 0 (Unassigned)*

The *CallAppearance* value that has been assigned to this call. For a specific device, the *CallAppearance* value is unique for all currently active calls.

CallAppearance values number from 1 up to the value set for the *MaxCallAppearances* property in the individual device class.

If an incoming call is received by PhoneX, the next free *CallAppearance* number is assigned before the *CallAlerting* event is fired. If the *MaxCallAppearences* has been exceeded, the *CallAppearance* parameter will be set to -1.

If a dial request (*CallDial*) is received with a *CallAppearance* value of 0, PhoneX automatically assigns the next free call appearance. If the received *CallAppearance* is not 0, PhoneX will not alter the value.

You can use this value to associate a call appearance indicator (Line Key) on your client application with a call class.

Note: This value is not necessarily the same as the actual call appearance on the physical station where the call appears.

CallDirection	*Long, Read-only*	*Default: 0*

This indicates whether the call was incoming to the monitored device or outgoing from the monitored device. For call direction values, refer to the enumeration *enCallDirection* in the *PhoneX Enumerations Guide*.

CalledDN	*String*	*Default: Empty*

The number dialled by the call. If this is an outbound call from the monitored device, this field will contain the destination number.

CalledName	*String, Read/write*	*Default: Empty*

The name of the person being called. This name is available only if the called party DN is also monitored by PhoneX. If the name is not available, this property contains the calledDN.

CallEndCause	*Long, Read-only*	*Default: -1*

Reason why the call ended. For call end cause values, refer to the enumeration *enEventCause* in the *PhoneX Enumerations Guide*.

CallerDigits *String, Read-only* *Default: Empty*

This contains any digits collected using the Definity ECS call prompting feature. For this feature to work, PhoneX must be monitoring the appropriate VDN devices.

CallerDN *String, Read/write* *Default: Empty*

The DN of the originator of the call. If the call is an outbound call from the monitored device, this field will contain the DN of the monitored device. If the call is inbound and the caller's number is not restricted, the caller's number will be presented in this field.

If the caller's number is marked as restricted and the PhoneX property to honor CLI restriction (*HonorDefinityCLIRestriction*) is True, this field will be replaced with the text taken from the *CLIRestrictedReplacementString* property.

See also *RestrictedCallerDN* property.

CallerName *String* *Default: Empty*

If the caller to the monitored device is from a device that is also monitored by PhoneX and PhoneX has collected a device name from the Definity ECS, this name will be placed in this field. If there is no device name or the device is not monitored, this field will be blank.

CallFlag1 *Variant, Read/write* *Default: Empty*

A free-form field that can be used by the host application. These fields are not modified by PhoneX.

CallFlag2 *Variant, Read/write* *Default: Empty*

A free-form field that can be used by the host application. These fields are not modified by PhoneX.

CallFlag3 *Variant, Read/write* *Default: Empty*

A free-form field that can be used by the host application. These fields are not modified by PhoneX.

CallFlag4 *Variant, Read / Write* *Default: Empty*

A free-form field that can be used by the host application. These fields are not modified by PhoneX.

CallID *Long, Read-only* *Default: 0*

The current Definity ECS-assigned call identifier for the call.

CallIdentifier *String, Read-only* *Default: Empty*

A string that uniquely identifies the call class within PhoneX. The call identifier takes the form of xxxx.yyyy.zzzz where:

xxxx is the primary monitored DN responsible for this call class.

yyyy is the current Definity CallID.

zzzz is a unique number generated by PhoneX .This number starts at 2000 and increments for each new call received to PhoneX.

CallModifiedMethod *Long, Read Only* *Default: 0*

This indicates the reason why the call class has changed when the *CallModified* or *ClassCallModified* events are received by the host application. For reason values, refer to the enumeration *enModifyCause* in the *PhoneX Enumerations Guide*.

CallStartCause *Long, Read-only* *Default: -1*

Reason why the call started. For call start cause values, refer to the enumeration *enEventCause* in the *PhoneX Enumerations Guide*.

CallState *Long, Read-only* *Default: 0*

The current state of the call. For call state values, refer to the enumeration *enCallState* in the *PhoneX Enumerations Guide*.

CallStatePrevious *Long, Read-only* *Default: 0*

The state the call was previously in. For call state values, refer to the enumeration *enCallState* in the *PhoneX Enumerations Guide*.

CallTimeAnswered *String, Read-only* *Default: Empty*

The time retrieved from the PC when the call associated with this call class is answered. If the call is not answered, this string will remain empty.

CallTimeEnd *String, Read-only* *Default: Empty*

The time retrieved from the PC when the call associated with this call class clears.

CallTimeStart *String, Read-only* *Default: Empty*

The time retrieved from the PC when the call class was created by PhoneX. If PhoneX did not create the call class, PhoneX will set this field when it first receives the class.

ConferencePendingState *Long, Read-only* *Default: 0*

This identifies if this call class is the primary device, that is, if it is the conferencing party or the secondary device the call is to be conferenced to. This enables alternation of call classes without losing the conference (alternating between two call appearances, each with a call class respectively).

For conference pending state values, refer to the enumeration *enConferencePendingState* in the *PhoneX Enumerations Guide*.

ConferencePendingType *Long, Read-only* *Default: 0*

This identifies the type of conference being performed and whether this call was part of the conference in progress. For conference type values, refer to the enumeration *enConferenceType* in the *PhoneX Enumerations Guide*.

DistributingDevice *String*

Specifies the ACD or VDN device that distributed the call to the station.

DN *String, Read-only* *Default: Empty*

The currently monitored device that generated this call class.

FriendlyName *String, Read/write* *Default: Empty*

String field provided for container application usage. PhoneX does not use this field. If the call class is created with PhoneX, this field will default to an empty string. This member stores a string that may be displayed to the application user.

LAIInfo

Look ahead information provided by the Definity ECS when calls are interflowed between switches.

MemberList *MemberListClass, Read-only* *Default: Default MemberList Class*

The *MemberList* is a property of type *MemberListClass* that contains information specific to the parties of a call. This includes the device numbers and their current talk and connection state.

Refer to Base Class Definition: *MemberList* Class.

NewCallClass *CallClass, Read-only* *Default: NULL (Nothing)*

The call class that this call class became. This field allows the call history to be viewed, forming a call chain.

Forward-looking call class link.

The validity of this information depends on the size of the call history list. Once the list size is reached, old call classes are automatically deleted. References to call classes that have been deleted are automatically deleted.

NewDN *String, Read/write* *Default: Empty*

This field will contain the new destination DN if the call is to be routed.

OCIInfo

Original call information.

OldCallClass *CallClass, Read-only* *Default: NULL (Nothing)*

The call class that was transformed into this call class by a call transfer or a call conference. This field allows the call history to be viewed, forming a call chain.

Backward-looking call class link.

The validity of this information depends on the size of the call history list. Once the list size is reached, old call classes are automatically deleted. References to call classes that have been deleted are automatically deleted. If a call class has been deleted from the Old Call List, this parameter will be set to NULL (Nothing).

OtherDN *String, Read/write* *Default: Empty*

Reserved. Used by AgentX to indicate the ‘other’ party (ie. the person you are connected to) in a two-party call. May be either the *CallerDN* or *CalledDN* depending on the *CallDirection* property.

Priority *Boolean, Read/write* *Default: False*

This field is used for outbound, on-switch (station-to-station) calls to invoke the Definity ECS priority feature.

Reason

Reason information. Event specific. For reason values, refer to the enumeration *enATTReason* in the *PhoneX Enumerations Guide*.

RestrictedCallerDN *String, Read/write* *Default: Empty*

This field will contain the same information as the CallerDN field.

Even if the caller’s number is marked as restricted and the PhoneX property to honor CLI restriction (*HonorDefinityCLIRestriction*) is set to True, the original value of the caller DN will display. It will not be replaced with the text taken from the *CLIRestrictedReplacementString* property.

ServiceObserved *Boolean, Read-only* *Default: False*

Service Observe property is True when the call is being service observed by another party. The observing party must have entered the call by dialing the Definity ECS service observe feature code.

If this call is the service observer call, this property will be False.

Note: This property is only valid for *StreamVersion* 6.

ServiceObserveDevice *String, Read-only* *Default: Empty*

The extension number of the party service observing this call. This party must have entered the call by dialing the Definity ECS service observe feature code. If the call is not being service observed, this property will be empty.

Note: This property is only valid for *StreamVersion* 6.

Split

Specifies the ACD split extension to which the call has been delivered.

TransferPendingState *Long, Read-only* *Default: 0*

This identifies if this call class is the primary device, that is, if it is the transferring party or the secondary device the call is transferred to. This enables alternation of call classes without losing the transfer (alternating between two call appearances, each with a call class respectively).

For transfer pending state values, refer to the enumeration *enTransferPendingState* in the *PhoneX Enumerations Guide*.

TransferPendingType *Long, Read-only* *Default: 0*

This identifies the type of transfer being performed and whether this call was part of the transfer in progress. For transfer type values, refer to the enumeration *enTransferType* in the *PhoneX Enumerations Guide.*

TrunkGroup *String, Read-only* *Default: Empty*

The Definity ECS trunk group the call was received on. This field requires the Avaya Computer Telephony *StreamVersion* with the Definity ECS to be 5 or higher. This field will not be present if caller information is present.

TrunkGroupMember *String, Read-only* *Default: Empty*

The Definity ECS trunk group member the call was received on. This field requires the Avaya Computer Telephony *StreamVersion* with the Definity ECS to be 5 or higher. This field will not be present if caller information is present.

UCID *String, Read-only* *Default: Empty*

The Universal Call Identifier is a parameter available with version 6 Definity ECS systems. It is a number unique to the call. This number is used in Avaya Call Management systems to uniquely identify the call. If the Definity ECS does not have this feature enabled, the contents will be empty.

UUI *String, Read/write* *Default: Empty*

For an inbound call, this field will contain any user-to-user information received with the call. If the call is outbound from the monitored device, the user-to-user information to be sent with the call will be in this field. (Making an outbound call using the *CallDial* method will take the information from this field to send with the call.)

The maximum length of user-to-user information currently accepted by the Definity ECS is 96 characters (assuming you have a Release 8 or better switch with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server; otherwise 32 characters for a switch prior to Release 8). A call class received with a UUI field that exceeds this will be rejected.

MemberList Class

The MemberList Class is a sub-class of CallClass. It contains a complete list of all the parties on the current call, their connection state and their talk state. The MemberList Class, which is found as a field in the CallClass, supports up to six parties on the call and gives a running count of the current number of parties.

MemberList Class Parameters

Class Parameter	Type	Default Value
ConnectionStateDN1	*long, Read-only*	*Default: 0*
The connection state of DN1 on this call. For connection state values, refer to the enumeration *enCallState* in the *PhoneX Enumerations Guide*.		
ConnectionStateDN2	*Long, Read-only*	*Default: 0*
The connection state of DN2 on this call. For connection state values, refer to the enumeration *enCallState* in the *PhoneX Enumerations Guide*.		
ConnectionStateDN3	*Long, Read-only*	*Default: 0*
The connection state of DN3 on this call. For connection state values, refer to the enumeration *enCallState* in the *PhoneX Enumerations Guide*.		
ConnectionStateDN4	*Long, Read-only*	*Default: 0*
The connection state of DN4 on this call. For connection state values, refer to the enumeration *enCallState* in the *PhoneX Enumerations Guide*.		
ConnectionStateDN5	*Long, Read-only*	*Default: 0*
The connection state of DN5 on this call. For connection state values, refer to the enumeration *enCallState* in the *PhoneX Enumerations Guide*.		
ConnectionStateDN6	*Long, Read-only*	*Default: 0*
The connection state of DN6 on this call. For connection state values, refer to the enumeration *enCallState* in the *PhoneX Enumerations Guide*.		
CurrentNumberOfMembers	*Long, Read-only*	*Default: 0*
The count of the current number of connection on this call.		
DN1	*String, Read-only*	*Default: Empty*
The identifier of the first party on the call.		
DN2	*String, Read-only*	*Default: Empty*
The identifier of the second party on the call.		
DN3	*String, Read-only*	*Default: Empty*
The identifier of the third party on the call.		
DN4	*String, Read-only*	*Default: Empty*
The identifier of the fourth party on the call.		
DN5	*String, Read-only*	*Default: Empty*
The identifier of the fifth party on the call.		
DN6	*String, Read-only*	*Default: Empty*

The identifier of the sixth party on the call.

TalkStateDN1 *Long, Read-only* *Default: 0*

The talk state of DN1 on this call. For talk state values, refer to the enumeration *enTalkState* in the *PhoneX Enumerations Guide*.

TalkStateDN2 *Long, Read-only* *Default: 0*

The talk state of DN2 on this call. For talk state values, refer to the enumeration *enTalkState* in the *PhoneX Enumerations Guide*.

TalkStateDN3 *long, Read-only* *Default: 0*

The talk state of DN3 on this call. For talk state values, refer to the enumeration *enTalkState* in the *PhoneX Enumerations Guide*.

TalkStateDN4 *long, Read-only* *Default: 0*

The talk state of DN4 on this call. For talk state values, refer to the enumeration *enTalkState* in the *PhoneX Enumerations Guide*.

TalkStateDN5 *long, Read-only* *Default: 0*

The talk state of DN5 on this call. For talk state values, refer to the enumeration *enTalkState* in the *PhoneX Enumerations Guide*.

TalkStateDN6 *long, Read-only* *Default: 0*

The talk state of DN6 on this call. For talk state values, refer to the enumeration *enTalkState* in the *PhoneX Enumerations Guide*.

AgentClass

The AgentClass contains detailed information about the agent session. This includes the agent ID and split/skill information.

When used in the EAS environment, PhoneX holds one AgentClass for a specific device. In the Non-EAS environment, PhoneX holds an AgentClass for each split the container application needs to log a device in for.

AgentClass validates the following parameters: *AgentPassword, AgentID, SplitSkill, AgentState, AgentMode, WorkMode* and *ReasonCode*. Any invalid input will not be accepted.

AgentClass Parameters

Class Parameter	Type	Default Value
AgentDN	*String, Read/write*	*Default: Empty*
The station number the agent is logging in with.		
AgentFlag1	*String, Read/write*	*Default: Empty*
A free-form string field that can be used by the host application. This field is not modified by PhoneX.		
AgentFlag2	*String, Read/write*	*Default: Empty*
A free-form string field that can be used by the host application. This field is not modified by PhoneX.		
AgentFlag3	*String, Read/write*	*Default: Empty*
A free-form string field that can be used by the host application. This field is not modified by PhoneX.		
AgentFlag4	*String, Read/write*	*Default: Empty*
A free-form string field that can be used by the host application. This field is not modified by PhoneX.		
AgentID	*String, Read/write*	*Default: Empty*
The ID code of the agent trying to log in.		
AgentIdentifier	*String, Read-only*	*Default: Empty*
A string that uniquely identifies the agent class within PhoneX. The agent identifier takes the form of xxxx.yyyy.zzzz where: xxxx is the monitored DN responsible for this agent class. yyyy is the agent ID for EAS, otherwise xxxx for Non-EAS. Reserved. zzzz is the split extension number for Non-EAS, otherwise blank for EAS. The AgentIdentifier field is a method inside the agent class that returns the current value. As a method, the values contained within the agent class can not be modified.		
AgentMode	*Long, Read/write*	*Default: -1*
The agent mode for the agent ID being monitored. For agent mode values, refer to the enumeration *enAgentMode* in the *PhoneX Enumerations Guide*.		

AgentPassword *String, Read/write* *Default: Empty*

The password of the agent ID trying to log in. If a password is not required, this parameter should be left blank.

AgentState *Long, Read/write* *Default: -1*

The agent state for the agent ID being monitored. For agent state values, refer to the enumeration *enAgentState* in the *PhoneX Enumerations Guide*.

AllowPendingStateChange *Boolean, Read/write* *Default: True*

By default, this will allow state changes of the agent mode and work mode while the user is still active on a call. The state changes will be placed on a pending state until such a time when the physical device is made idle. When the physical device is idle, the state will change to that as specified by the pending states.

FriendlyName *String, Read/write* *Default: Empty*

A free-form string field that can be used by the host application. This field is not modified by PhoneX.

PendingReasonCode *EnReasonCode, Read/write* *Default: 0*

This will hold the reason code to use when the state changes when the physical device is idle.

PendingWorkMode *EnAgentWorkMode, Read/write* *Default: -1*

This will hold the new work mode to use when the state changes when the physical device is idle.

ReasonCode *Long, Read/write* *Default: 0*

The reason code used to support the change of state. This parameter is only used in a change of state from 'anything' to Auxiliary mode or an attempt to log out. The values must be between 0 and 9.

SplitSkill *String, Read/write* *Default: Empty*

The split skill number (hunt group DN) the agent will be logging in to. If the system is operating in EAS mode, this field is left blank.

TalkState

Indicates if the agent is idle (ready to accept calls) or busy (occupied with serving a call).

WorkMode *Long, Read/write* *Default: 3*

The work mode for the agent ID being monitored. For agent work mode values, refer to the enumeration *enAgentWorkMode* in the *PhoneX Enumerations Guide*.

DeviceClass

The DeviceClass contains all the information about an individual device as seen from the perspective of an individual device. This means that PhoneX can contain more than one device to be monitored. The available types of device that may be monitored are station, VDN and split/skills.

DeviceClass Parameters

Class Parameter	Type	Default Value
ActiveCallClasses		
A collection of calls currently at this device.		
AgentClasses		
A collection of agent classes for this device.		
CallCountCurrent	*Long, Read-only*	*Default: 0*
A counter of the number of calls currently active on the device. This includes calls alerting at the device and those on hold at the device.		
CallCountTotal	*Long, Read-only*	*Default: 0*
A count of the total number of calls that have been (or are) active at the device. This includes inbound calls that were not answered or went to cover.		
ClipBoardAutoPasteFormat	*String*	
The format for information to be pasted into the clipboard when a new call arrives for this device. If the string is blank, no data will be pasted. Both literals and place holders can be used. U=UUI ?=CLI/ANI D=Collected Digits N=Caller Name		
DeviceDN	*String, Read/write*	*Default: Empty*
The number of the device being monitored by PhoneX.		
DeviceFlag1	*String, Read/write*	*Default: Empty*
A free-form string field that can be used by the host application. This field is not modified by PhoneX.		
DeviceFlag2	*String, Read/write*	*Default: Empty*
A free-form string field that can be used by the host application. This field is not modified by PhoneX.		
DeviceFlag3	*String, Read/write*	*Default: Empty*
A free-form string field that can be used by the host application. This field is not modified by PhoneX.		
DeviceFlag4	*String, Read/write*	*Default: Empty*
A free-form string field that can be used by the host application. This field is not modified by PhoneX.		
DeviceIdentifier	*String, Read-only*	*Default: Empty*

A string that uniquely identifies the device class within PhoneX. The *DeviceIdentifier* takes the form of xxxx.yyyy where:

xxxx primary or secondary monitored DN responsible for this device class

yyyy is a unique number generated by PhoneX that starts at 2000. It increments for each new device under monitor by PhoneX.

DeviceState	*Long, Read-only*	*Default: 0*

The current state of the device. For device state values, refer to the enumeration *enPrimaryDNState* in the *PhoneX Enumerations Guide*.

DeviceStatePrevious	*Long, Read-only*	*Default: 0*

The previous state of the device. For device state values, refer to the enumeration *enPrimaryDNState* in the *PhoneX Enumerations Guide*.

DeviceType	*Long, Read-only*	*Default: -1*

The type of device this class is controlling. For device type values, refer to the enumeration *enDeviceType* in the *PhoneX Enumerations Guide*.

FriendlyName	*String, Read/write*	*Default: Empty*

A free-form string field that can be used by the host application. This field is not modified by PhoneX.

MaxCallAppearances	*Long, Read/write*	*Default: 3*

This indicates the number of active calls that are expected for the device in question. When a call alerts to a device, it is assigned the next free call appearance. If the next call appearance exceeds the *MaxCallAppearance* property, the call class call appearance is set to -1.

MonitorType	*Long, Read/write*	*Default: 0*

The type of monitor this device is set for. For monitor type values, refer to the enumeration *enMonitorType* in the *PhoneX Enumerations Guide*.

MWTApplication	*Long, Read-only*	*Default: 0*

This indicates the applications that have set the Message Waiting feature. For MWT application values, refer to the enumeration *enMWTApplication* in the *PhoneX Enumerations Guide*.

NextCallAppearance	*Long, Read-only*	*Default: 0*

The next call appearance that is linked to the current call appearance.

Note: This may not match the call appearance selected by the physical device.

NotifyType	*Long, Read-only*	*Default: 0*

Reserved. Not used.

OldCallClasses

A collection of old calls for this device.

QueryFWDStatus	*Boolean, Read/write*	*Default: False*

If set to True, PhoneX will automatically poll the Definity ECS to determine the status of the Call Forward feature.

QueryMWTStatus	*Boolean, Read/write*	*Default: False*

If set to True, PhoneX will automatically poll the Definity ECS to determine the status of the Message Waiting feature.

QuerySACStatus	*Boolean, Read/write*	*Default: False*

If set to True, PhoneX will automatically poll the Definity ECS to determine the status of the Send All Calls feature.

SelectedCallAppearance *Long, Read-only* *Default: 0*

The call appearance currently selected by the application.

Note: This may not match the call appearance selected by the physical device.

StatusFWD *String, Read/write* *Default: Empty*

The current status of the device Call Forward parameter. This field is only valid if the device is a station. If the device being monitored has the Call Forward feature activated, this string will contain the forward destination. When empty (zero length) the device is not forwarded.

StatusMWT *Boolean, Read/write* *Default: False*

The current status of the device Message Waiting parameter. This field is only valid if the device is a station. When set, the device has a message set.

StatusSAC *Boolean, Read/write* *Default: False*

The current status of the Send All Calls parameter. This field is only valid if the device is a station. When set, the device has the Send All Calls feature activated.

SwitchName *String, Read/write* *Default: Empty*

The name associated with the device retrieved from the Definity ECS. For this to be valid, the link to the Telephony Server must be *StreamVersion* 5 or higher.

Error Class

The ErrorClass contains detailed information about the error that has occurred. This includes the error code, type, level information etc.

PhoneX will issue a *TSError* event whenever a failure has occurred in PhoneX.

ErrorClass Parameters

Class Parameter	Type	Default Value
DisplayError	*Long, Read/write*	*Default: 0*
Reserved. Not used.		
ErrorCode	*Long, Read/write*	*Default: 0*
The error value that is returned.		
ErrorContext	*Long, Read/write*	*Default: 0*
Reserved. Not used.		
ErrorDevice	*String, Read/write*	*Default: Empty*
The device that was monitored and that caused the error to happen.		
ErrorDeviceType	*Long, Read/write*	*Default: 0*
Reserved. Not used.		
ErrorLevel	*EnErrorLevel, Read/write*	*Default: 0*
This specifies the severity of the error.		
E*rrorType*	*EnErrorType, Read/write*	*Default: 0*
This specifies the type of error and enables the user to investigate exactly how the error occurs.		
InvokeID	*Long, Read/write*	*Default: 0*
The *invokeID* that caused the error to occur.		
ResourceTagError	*Long, Read/write*	*Default: 0*
The language-dependant error information associated with the *ErrorCode*.		
ResourceTagResolution	*Long, Read/write*	*Default: 0*
The language-dependant resolution for the *ErrorCode* that occurred.		

LAIInformation

For information on LAIInformation, refer to the *Programmer's Guide for Definity Enterprise Communications Server (Defprog.pdf)* on the Avaya Computer Telephony CD-ROM.

OCIInformation

For information on OCIInformation, refer to the *Programmer's Guide for Definity Enterprise Communications Server (Defprog.pdf)* on the Avaya Computer Telephony CD-ROM.

StringCollection

StringCollection is ? that contains detailed information about ?

StringCollection Parameters

Class Parameter	Type	Default Value
Add	*Long*	
Count	*String*	
Item	*String*	
Remove		

RuntimeLicenseInformation

RuntimeLicenseInformation is an object of PhoneXLicenseInfo class. It contains information regarding the current runtime license.

PhoneXLicenseInfo Parameters

Class Parameter	Type	Default Value
LicenseHolder	*String*	

The name of the company/user that owns the runtime license key.

LicenseNumber	*Long*	

The license number that has been issued by the License Server.

LicenseTotalPurchased	*Long*	

The total amount of purchased licenses.

LicenseType	Long	

The type of license that has been issued.

ActiveCallClasses Class

The ActiveCallClasses collection is of type CallClassListActive. ActiveCallClasses will be able to retrieve an active CallClass object using the methods and properties exposed by the CallClassListActive base collection class. Once a call is released, PhoneX removes the associated CallClass from the collection and places it into the OldCallClasses collection.

CallClassListActive Parameters

Class Parameter	Type	Syntax
Add	*Method*	*Add() As CallClass*

Adds a *CallClass* to the collection. Sample code:

```
Dim cls as CallClass
Set cls = Me.px.ActiveCallClasses.Add()
 cls.CalledDN = "8888"
 cls.CallerDN = "8575"
 cls.UUI = "Hello"
Me.px.CallDial cls
```

Count	*Long*	

Returns the number of *CallClass* that is stored within this collection class. Sample code:

```
Dim I as Integer
Dim cls as CallClass
For I = 1 to Me.px.ActiveCallClasses.Count
      Set cls = Me.px.ActiveCallClasses.Item(I)
      If Len(Trim(cls.UUI)) > 0 Then
      DoStatus cls.UUI
      End If
Next I
```

Item	*Method*	*Item(Index) As CallClass*

Retrieves a particular *CallClass* from the collection. Note: PhoneX will handle both the *Index* value as the position in the list (1-based index) or the *CallIdentifier* of the call class. Sample code:

```
'List1 is the list that contains the call identifiers
Dim cls as CallClass
'Get the first call identifier in List1
If List1.List(0) <> "" Then
      Set cls = 
Me.px.ActiveCallClasses.Item(List1.List(0))
End If
```

ItemActiveCall	*Method*	*ItemActiveCall(DN As String) As CallClass*

Retrieves a particular *CallClass* of the specified DN that is currently active from the collection. Sample code:

```
'Assuming we have an active call on DN 8575

Dim cls As CallClass

Set cls = me.px.ActiveCallClasses.ItemActiveCall("8575")

If Not cls is Nothing Then

      Me.px.CallHold(cls)

End If
```

ItemCallByCallAppearance	*Method*	*ItemCallByCallAppearance(DN As String, CallAppearance As Long) As CallClass*

Retrieves a particular *CallClass* of the specified DN on a specified call appearance from the collection. It will return NULL if no call classes are associated with the call appearance for the specified DN.

Sample code:

```
'Assuming we have an active call on DN 8575

Dim cls As CallClass

Set cls = me. _
      px.ActiveCallClasses.ItemCallByCallAppearance("8
575",1)

If Not cls is Nothing Then

      Me.px.CallUnHold(cls)

End If
```

Remove	*Method*	*Remove(Index)*

Removes a particular *CallClass* from the collection. Note: PhoneX will handle both the *Index* value as the position in the list (1-based index) or the *CallIdentifier* of the call class. Sample code:

```
'List1 is the list that contains the call identifiers

'Remove the Call Class with first call identifier in List1

If List1.List(0) <> "" Then

      Me.px.ActiveCallClasses.Remove(List1.List(0))

End If
```

OldCallClasses Class

The OldCallClasses is of type CallClassListOld. It contains the collection of CallClass objects that are no longer part of an active call. These are stored for historical purposes and can be removed manually by using the *Remove* method. PhoneX will automatically replace the oldest CallClass once the list size reaches *MaxOldCallListSize*.

Note: The OldCallClasses exists in memory as long as the instance of PhoneX is still running.

CallClassListOld Parameters

Class Parameter	Type	Syntax
Count	*Long*	

Returns the number of *CallClass* that is stored within this collection class. Sample Code:

```
Dim I as Integer
Dim cls as CallClass
For I = 1 to Me.px.OldCallClasses.Count
      Set cls = Me.px.OldCallClasses.Item(I)
      If Len(Trim(cls.UUI)) > 0 Then
      DoStatus cls.UUI
      End If
Next I
```

Item	*Method*	*Syntax*	*Item(Index) As CallClass*

Retrieves a particular *CallClass* from the collection. Note: PhoneX will handle both the *Index* value as the position in the list (1-based index) or the *CallIdentifier* of the call class.

Sample Code:

```
'List1 is the list that contains the call identifiers
Dim cls as CallClass
'Get the first call identifier in List1
If List1.List(0) <> "" Then
      Set cls = Me.px.OldCallClasses.Item(List1.List(0))
End If
```

Remove	*Method*	*Remove(Index) As CallClass*

Removes a *CallClass* from the collection. Sample Code:

```
Dim I as Integer
For I = 1 to Me.px.OldCallClasses.Count
      Me.px.OldCallClasses.Remove(I)
Next I
```

AgentClasses Class

The AgentClasses Class is of type AgentClassList collection class that contains all the class information of logged-in agents. AgentClasses stores the agent classes in a manner of an unsorted list. Hence the newest AgentClass object is added to the bottom of the collection.

The collection class exposes the *Add* method for adding new agent classes to the list and also for retrieval by using the *Item* method. The *Count* property allows the user to check how many agents are currently logged in using PhoneX. Any agents that log out will be removed from this list.

AgentClassList Parameters

Class Parameter	Type	Syntax
Add	*Method*	*Add()*

Adds an *AgentClass* to the collection. Sample code:

```
Dim agt as AgentClass
Set agt = Me.px.AgentClasses.Add()
agt.AgentDN = "8575"
agt.AgentID = "9809"
agt.AgentPassword = "Agent1"
Me.px.AgentClasses.Add(agt)
```

Class Parameter	Type	Syntax
Count	*Long*	

Returns the number of *AgentClass* stored within this collection class. Sample code:

```
Dim I as Integer
Dim agt as AgentClass
For I = 1 to Me.px.AgentClasses.Count
      Set agt = Me.px.AgentClasses.Item(I)
      If agt.AgentMode = amNotReady And agt.WorkMode =
wmAUX Then
      DoStatus "AUX Mode"
      ElseIf agt.AgentMode = amWorkNotReady And
agt.WorkMode = _  wmACW Then
      DoStatus "ACW Mode"
      ElseIf agt.AgentMode = amReady And agt.WorkMode =
wmAutoIn Then

      DoStatus "AutoIn Mode"

      ElseIf agt.AgentMode = amReady And agt.WorkMode =
wmManualIn _      Then

      DoStatus "ManualIn Mode"

      End If

Next I
```

Item	*Method*	*Item(Index) As AgentClass*

Retrieves a particular *AgentClass* from the collection. Note: The *Index* can receive either the position in the list (1-based index) or the *AgentIdentifier* of the agent class. Sample code:

```
'List1 is the list that contains the agent identifiers

Dim agt as AgentClass

'Get the first agent identifier in List1

If List1.List(0) <> "" Then

      Set agt = Me.px.AgentClasses.Item(List1.List(0))

End If
```

DeviceClasses Class

The DeviceClasses collection class is of type DeviceClassList collection class that contains all the class information of devices that are currently monitored by PhoneX. The collection class stores the device classes in a manner of an unsorted list. Hence the newest DeviceClass object that is monitored successfully will be added to the bottom of the collection.

It exposes the *Add* method for adding new device classes that are successfully being monitored to the list and also for retrieval by using the *Item* method. The *Count* property allows the user to check how many devices PhoneX is currently monitoring.

DeviceClassList Parameters

Class Parameter	Type	Syntax
Add	*Method*	*Add()*

Adds a *DeviceClass* to the collection. Sample Code:

```
Dim dev as DeviceClass
Set dev = Me.px.DeviceClasses.Add()
dev.DeviceDN = "8575"
dev.MonitorType = mtCompleteMonitor
Me.px.DeviceClasses.Add(dev)
```

Count	*Long*	

Returns the number of *DeviceClass* that is stored within this collection class.

Sample Code:

```
Dim I as Integer
Dim dev as DeviceClass
For I = 1 to Me.px.DeviceClasses.Count
      Set dev = Me.px.DeviceClasses.Item(I)
      If Not dev Is Nothing Then
      DoStatus dev.DeviceDN
      End If
Next I
```

Item	*Method*	*Item(Index) As DeviceClass*

Retrieves a particular *DeviceClass* from the collection. Note: The index can receive either the position in the list (1-based index) or the device identifier of the device class.

Sample Code:

```
Dim dev As DeviceClass

Set dev = Me.px.DeviceClasses.Item(1)

If Not dev Is Nothing Then

      DoStatus "Monitored Device = "+ dev.DeviceDN

End If
```

TServers Class

The TServers collection class is of type TServerList collection class that contains all the strings of telephony link names. The collection class stores these strings in a manner of an unsorted list. Hence the newest TServer link name that PhoneX locates will be added to the bottom of the collection.

It exposes the *Item* method for retrieval of a required string. The *Count* property allows the user to check how many telephony links there are which PhoneX was able to locate.

TServerList Class Parameters

Class Parameter	Type	Syntax
Count	*Long*	

Returns the number of telephony link names currently stored within this collection class.

Sample Code:

```
Dim I as Integer
For I = 1 to Me.px.TServers.Count
      DoStatus Me.px.TServers.Item(I)
Next I
```

Class Parameter	Type	Syntax
Item	*Method*	*Item(Index) As String*

Retrieves a particular telephony link name from the collection. Note: The *Index* can only receive the position in the list (one-based index).

Sample Code: (see above for details)

Use of Class Information

For methods and events exposed by Contact Center Express components, information is passed between the component and the controlling application using one or more of the base classes. It is the intention of these components that class information is held by the base component PhoneX. It is unnecessary and highly undesirable that applications using these controls should hold copies of any class variable. To facilitate this operation, methods are provided in the controls to allow class variables to be created and stored in the base component, and to retrieve these classes from PhoneX. These are done by means of using classes that reference their respective collection classes. The methods and properties exposed by the collection class can be used by the class that references it. These methods will also ensure that the class variables are initialized to the correct default values. If these methods are not used, the base components will attempt to add new classes to the class list. It should be noted that some of the collection classes do not facilitate adding and/or removing classes as they are Read-only.

Creating a New Class

All classes are created and held within PhoneX. Some collection classes have an *Add()* method to allow an application to create a new class. Refer to the Class Structures section for more information.

Retrieving an Existing Class

Almost all events return a valid class as part of the event parameters. Most methods also require the passing of a valid class in order to perform the action on the specified call, device or agent.

Each class is identified by a unique token, called the identifier. In the case of a call class, this identifier is called the *CallIdentifier*, and in the case of an agent class, the *AgentIdentifier*. These identifiers are generated by PhoneX as the class is created, and are used by all components (including your application) to retrieve the required class through the Index parameter of the Item methods. It is also possible to retrieve the required class if the index number is known.

CallClass: CallIdentifier

To retrieve a call class, the *CallIdentifier* is required. It is left up to the user to decide whether to store the *CallIdentifier* internally or to use the call appearance as the basis for retrieving the *CallIdentifier*. For example, upon the *CallAlerting()* event firing, a valid call class for the alerting call is supplied. Within the call class is the *CallIdentifier* property. When you wish to retrieve information about the call (perhaps to place it on hold or transfer it), use the *ActiveCallClasses.Item(x)* function and supply the internally stored *CallIdentifier* as the parameter for x or if the index number is known, use that number in place of x. Two further methods are available for retrieving the call classes. These are the *ItemActiveCall* and the *ItemCallByCallAppearance* methods. Both methods allow the user to enter the monitored DN as its first parameter and the latter allows for the call appearance number for retrieval.

AgentClass: AgentIdentifier

To retrieve an agent class, your application must store the *AgentIdentifier* internally. See the call class *CallIdentifier* for more information.

CHAPTER 4

OCX Class Control Events

This chapter contains the events that return from PhoneX whenever there is an update performed on the classes within PhoneX.

In This Chapter

ClassCallModified

Syntax:	*ClassCallModified(ByVal clsCall As CallClass, ByVal Reason As Long)*
Description:	Fires when the OCX control has made a change to a call class that is not a result of call control methods being invoked or events being received from the Telephony Server.
Response to method:	[*GetAllCallClass*]

Parameters

clsCall	The call class that has been modified.
Reason	The reason this event has fired. For reason values, refer to the enumeration *enClassModifyCause* in the *PhoneX Enumerations Guide*.

Class Settings

None.

Sample Code

```
Private Sub px_ClassCallModified(ByVal clsCall As
CallClass, ByVal Reason As Long)

      Dim OurCallClass As CallClass

      ‘Retrieve the call classes for use in the code.

      If Reason = MOD_QUERY_REQUEST Then

            OurCallClass = clsCall

      End If

End Sub
```

ClassDeviceModified

Syntax:	*ClassDeviceModified(ByVal clsDevice As DeviceClass, ByVal Reason As Long)*
Description:	Fires when the OCX control has made a change to a device class that is not a result of device control methods being invoked or events being received from the Telephony Server.
Response to method:	[*GetAllDeviceClass*]

Parameters

clsDevice	The device class that has been modified.

Reason	The reason this event has fired. For reason values, refer to the enumeration *enClassModifyCause* in the *PhoneX Enumerations Guide*.

Class Settings

None.

Sample Code

```
Private Sub px_ClassDeviceModified(ByVal _
clsDevice As DeviceClass, ByVal Reason As Long)
      'Retrieve this device class
      If Reason = MOD_QUERY_REQUEST Then
      clsDevice.StatusSAC = False
      clsDevice.StatusMWT = False
      End If
End Sub
```

ClassAgentModified

Syntax:	*ClassAgentModified(ByVal clsAgent As AgentClass, ByVal Reason As Long)*
Description:	Fires when the OCX control has made a change to an agent class that is not a result of call or agent control methods being invoked or events being received from the Telephony Server.
Response to method:	[*AgentClasses.Add*]

Parameters

clsAgent	The agent class that has been modified.
Reason	The reason this event has fired. For reason values, refer to the enumeration *enClassModifyCause* in the *PhoneX Enumerations Guide*.

Class Settings

None.

Sample Code

```
Private Sub px_ClassAgentModified(ByVal clsAgent As
AgentClass, ByVal Reason As Long)

      If Reason = MOD_NEWAGENT Then

      'Change the work mode to after call work mode:

      clsAgent.WorkMode = enAgentWorkMode.wmACW

      End If

End Sub
```

CHAPTER 5

OCX Link Control Properties

This chapter contains the property variables of PhoneX that relate to the telephony link.

In This Chapter

TServerLinkName/TServerLinkName Secondary

Syntax:	*TserverLinkName As String*
	TServerLinkNameSecondary As String
Description:	The *TServerLinkName* and *TServerLinkNameSecondary* properties are set to the name of the Telephony Server telephony link name (eg. AVAYA#G3_SWITCH#CSTA#TSERVER01).

Usage Notes

The properties are string properties. The default values are empty strings. The *TserverLinkNameSecondary* is for a hot standby link to a standby server that will take over if the primary server fails.

TServerUserName/TServerUserNam eSecondary

Syntax:	*TserverUserName As String*
	TserverUserNameSecondary As String
Description:	The *TServerUserName* and *TServerUserNameSecondary* properties are set to the name of the Telephony Server user ID.

Usage Notes

These properties are string type properties. The default values are empty strings. These shall be filled with a valid Telephony Server user ID. The *TserverUserNameSecondary* is for a hot standby user name to a standby server that will take over if the primary server fails.

TServerUserPassword/TServerUser PasswordSecondary

Syntax:	*TserverUserPassword As String*
	TServerUserPasswordSecondary As String
Description:	The *TServerUserPassword* and *TServerUserPasswordSecondary* properties are set to the password for the name of the Telephony Server user ID used.

Usage Notes

These properties are of type string. The default values are empty strings. These shall be filled with a valid Telephony Server user password associated with a valid Telephony Server user ID. The *TserverUserPasswordSecondary* is for a hot standby user password to a standby server that will take over if the primary server fails.

PhoneXEnabled

Syntax:	*PhoneXEnabled As Boolean*
Description:	Setting this property to True will cause PhoneX to log in the user with the given password and telephony link to the Telephony Server. This will enable computer telephony integration.

Usage Notes

The property is a Boolean-type property. The default value is False. When set to True, it will cause PhoneX to log the user into the Telephony Server with the specified user ID, password and T-link. If login is successful, PhoneX will issue the *TSLoggedIn* event. If login fails, PhoneX will issue a *TSError* event with the appropriate error codes.

This property is valid only in the runtime environment and should be set after the *TserverLinkName*, *TserverUserName* and *TserverUserPassword* properties have been set.

Setting this property to False in the runtime environment will result in any open links to the Telephony Server being closed.

ProvideEventsForLinkRecovery

Syntax:	*ProvideEventsForLinkRecovery As Boolean*
Description:	When set to True, PhoneX provides the controlling application with events resulting from the failure of the primary Telephony Server connection.

Usage Notes

The property is a Boolean-type property. The default value is True.

When set to True, PhoneX will provide the controlling application with events indicating that the primary telephony link has failed. PhoneX will then attempt to activate the secondary link.

When set to False, PhoneX will attempt to activate the secondary link but no events will be returned to the controlling application stating that the primary link has failed.

AutoFallBackToPrimaryServer

Syntax:	*AutoFallBackToPrimaryServer As Boolean*
Description:	Causes PhoneX to automatically attempt to activate the primary server after it has failed.

Usage Notes

The property is a Boolean-type property. The default value is True.

When set to True, PhoneX automatically tries to activate the primary server after it has failed. If the primary server fails and the secondary server is activated, PhoneX starts to monitor the primary server, waiting for it to be brought back online. Once online, PhoneX tries to switch its active link back to the primary server. This transition is governed by the *AutoFallBackToPrimaryServerTime* property or if the secondary server fails.

If the property is set to False, PhoneX remains connected to the secondary server and continues to use the secondary server until the application is restarted. If the secondary server also fails, the application will lose telephony functionality.

AutoFallBackToPrimaryServerTime

Syntax:	*AutoFallBackToPrimaryServerTime As Long*
Description:	Specifies how long PhoneX will wait before switching back to the primary server after the server is back online.

Usage Notes

This property has a default value set at 1 minute. The range is 1-10 minutes. *AutoFallBackToPrimaryServerTime* specifies the time period, in minutes, that PhoneX will wait after determining that the primary server is online before switching control back to the primary server.

If the *AutoFallBackToPrimaryServer* property is set to False, the *AutoFallBackToPrimaryServerTime* property has no effect.

ActiveTServerLink

Syntax:	*ActiveTServerLink As enActiveServer*
Description:	Specifies the Telephony Server link being used as the active connection.

Usage Notes

This allows other applications to be aware of the current active Telephony Server link being used. If there are no active Telephony Server links, the application should stop all telephony functionality. For active server values, refer to the enumeration *enActiveServer* in the *PhoneX Enumerations Guide*.

CHAPTER 6

OCX Link Control Methods

This chapter contains the relevant PhoneX methods that relate to the telephony link and monitoring.

In This Chapter

TSListServers

Syntax:	*TSListServers(ByVal MaxListCount As Long) As Long*
Description:	Requests a list of available Telephony Servers.
Returns:	Long

Parameters

MaxListCount	Maximum number of servers to find before ending the search.

Return Values

pxErrorUnknown	The request has failed due to an unknown network problem.
pxNoServer	The request has failed because the client is using TCP/IP and the IP addresses are not set correctly.

Usage Notes

The *TSListServers* interrogates the TSAPI environment and returns, via the *TSServerAvailable* event, the names of the Telephony Servers found on the network. The *MaxListCount* variable specifies the maximum number of servers to find. If this parameter is set to 0, all servers will be listed.

Return Events (in order fired)

TSServerAvailable	This returns with the list of servers available for telephony integration. These servers would have Avaya Computer Telephony running in order to be returned with this event.

Sample Code

```
Private Sub cmdTSListServers_Click()
      Dim lRtn As Long

      ‘Lists 10 telephony servers available on the network
      lRtn = px.TSListServers(10)
End Sub
```

TSShutDown

Syntax:	*TSShutDown() As Long*
Description:	Closes the stream open to the Telephony Server and cancels all monitor and routing requests. The *IsAvailable* property is updated to reflect this change.
Returns:	Long

Parameters

None.

Return Values

pxNoError	The method completed successfully. The link has been closed to the Telephony Server.
pxBadHandle	There is no connection open to a Telephony Server.

Usage Notes

All memory is released and the current and old call lists purged of contents. No call or device control methods can be called once this method has returned.

Sample Code

```
Private Sub cmdTSShutDown_Click()
      Dim lRtn As Long

      ‘Stop monitoring all devices
      ‘Close the telephony link
      lRtn = px.TSShutDown()
End Sub
```

TSMonitorStation

Syntax:	*TSMonitorStation(ByVal clsDevice As DeviceClass) As Long*
Description:	Requests a station DN be monitored and specifies the type(s) of monitoring required.
Returns:	Long

Parameters

clsDevice	The device class containing the information about the device to be monitored.

Return Values

pxNoError	The method completed successfully. The device has been monitored successfully.
pxClassEmpty	The *clsDevice* passed with this method does not point to a valid class object.
pxInvalidClass	The *MonitorType* option is set to monitor stop but the class object, *clsDevice*, is not known to PhoneX.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *DeviceDN* exceeds the maximum number of character (64) or is zero length or the *MonitorType* parameter contains an invalid setting.

pxStreamFailed	The link to the Telephony Server has failed.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Usage Notes

The *TSMonitorStation* initiates the monitor on the specified device DN. The monitor options allow for the device to be fully monitored, monitored as a busy lamp indicator or unmonitored.

If the device class does not exist within the PhoneX class collection, PhoneX will add it and make it available. If the *TSMonitorStation* fails (eg. invalid device or insufficient user permissions), the error will be reported through the *TSError* mechanism (including the *invokeID*). Once this error has been reported, the device class will be removed from the PhoneX list. Once all clients have removed references from it, the device class will be destroyed.

If the monitor of the device is successful, this will be reported in the *TSMonitorStationReturn* event. If the Avaya Computer Telephony *StreamVersion* (see *TSLoggedIn* event) is a value of 5 or above, PhoneX will issue a *ClassDeviceModified* event at some interval after the *TSMonitorStationReturn* event. At this time, certain parameters will have been updated in the device class (eg. *SwitchName*).

Return Events (in order fired)

TSMonitorStationReturn	This event returns when the monitoring of the specified DN is successful.

Class Settings

DeviceDN	The DN (extension number) of the Definity/Multivantage device to be monitored.
MonitorType	The type of monitoring to be performed on the specified station. For monitor type values, refer to the enumeration *enMonitorType* in the *PhoneX Enumerations Guide*.

Sample Code

```
Private Sub cmdTSMonitorStation_Click()
      Dim clsDev as DeviceClass
      Dim lRtn As Long

      Set clsDev = px.DeviceClasses.Add()
      If Not clsDev Is Nothing Then
            ‘Initialize new device class with desired
info.With the device class information, monitor it
            lRtn = px.TSMonitorStation(clsDev)
      End If
End Sub
```

TSMonitorSkill

Syntax:	*TSMonitorSkill(ByVal clsDevice As DeviceClass) As Long*
Description:	Requests a split or skill be monitored and specifies the type(s) of monitoring required.
Returns:	Long

Parameters

clsDevice	The device class containing the information about the device to be monitored

Return Values

pxNoError	The split or skill was monitored successfully.
pxClassEmpty	The *clsDevice* passed with this method does not point to a valid class object.
pxInvalidClass	The *MonitorType* option is set to monitor stop but the class object, *clsDevice*, is not known to PhoneX.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *DeviceDN* exceeds the maximum number of character (64) or is zero length or the *MonitorType* parameter contains an invalid setting.
pxStreamFailed	The link to the Telephony Server has failed.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Usage Notes

The *TSMonitorSkill* initiates the monitor on the specified skill. The monitor options allow for the device to be fully monitored, and receive information about all calls presented to the skill. Monitoring can also be achieved for agent login and logout events for the specified skill.

If the device class does not exist within the PhoneX class collection, PhoneX will add it and make it available. If the *TSMonitorStation* fails (eg. invalid device or insufficient user permissions), the error will be reported through the *TSError* mechanism (including the *invokeID*). Once this error has been reported, the device class will be removed from the PhoneX list. Once all clients have removed references from it, the device class will be destroyed.

If the monitor of the device is successful, this will be reported in the *TSMonitorSkillReturn* event. If the Avaya Computer Telephony *StreamVersion* (see *TSLoggedIn* event) is a value of 5 or above, PhoneX will issue a *ClassDeviceModified* event at some interval after the *TSMonitorSkillReturn* event. At this time, certain parameters will have been updated in the device class (eg. *SwitchName*).

Return Events (in order fired)

TSMonitorSkillReturn	This event returns when the monitoring of the specified skill is successful.

Class Settings

DeviceDN	The DN (extension number) of the Definity/MultiVantage device to be monitored.
MonitorType	The type of monitoring to be performed on the specified skill. For monitor type values, refer to the enumeration *enMonitorType* in the *PhoneX Enumerations Guide*.

Sample Code

See example for *TSMonitorStation*.

TSMonitorVDN

Syntax:	*TSMonitorVDN(ByVal clsDevice As DeviceClass) As Long*
Description:	Requests a VDN be monitored and specifies the type(s) of monitoring required.
Returns:	Long

Parameters

clsDevice	The device class containing the information about the device to be monitored.

Return Values

pxNoError	The method completed successfully. The VDN is monitored successfully.
pxClassEmpty	The *clsDevice* passed with this method does not point to a valid class object.
pxInvalidClass	The *MonitorType* option is set to monitor stop but the class object, *clsDevice*, is not known to PhoneX.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *DeviceDN* exceeds the maximum number of character (64) or is zero length or the *MonitorType* parameter contains an invalid setting.
pxStreamFailed	The link to the Telephony Server has failed.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Usage Notes

The *TSMonitorVDN* initiates the monitor on the specified VDN. Monitor options allow for the device to be fully monitored or unmonitored only.

If the Device class does not exist within the PhoneX class collection, PhoneX will add it and make it available.

If the *TSMonitorStation* fails (eg. invalid device or insufficient user permissions), the error will be reported through the *TSError* mechanism (including the *invokeID*). Once this error has been reported, the device class will be removed from the PhoneX list. Once all clients have removed references from it, the device class will be destroyed.

If the monitor of the device is successful, this will be reported in the *TSMonitorVDNReturn* event. If the Avaya Computer Telephony *StreamVersion* (see *TSLoggedIn* event) is a value of 5 or above, PhoneX will issue a *ClassDeviceModified* event at some interval after the *TSMonitorVDNReturn* event. At this time, certain parameters will have been updated in the device class (eg. *SwitchName).*

Return Events (in order fired)

TSMonitorVDNReturn	This event returns when the VDN was successfully monitored.

Class Settings

DeviceDN	The DN (extension number) of the Definity/MultiVantage device to be monitored.
MonitorType	The type of monitoring to be performed on the specified VDN. For monitor type values, refer to the enumeration *enMonitorType* in the *PhoneX Enumerations Guide.*

Sample Code

See example for *TSMonitorStation.*

TSGetAuthorizationType

Syntax:	*TSGetAuthorizationType(ByVal ServerName As String) As Long*
Description:	Determines the login and password requirements when opening a telephony stream for the advertised service. It determines whether the user that is logged on to the PC needs to supply a password to be able to use telephony or whether authentication is required again.
Returns:	Long

Parameters

ServerName	The string that contains the telephony link name, eg. AVAYA#G3_SWITCH#CSTA#SERVER

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

TSAuthorizationType	This event indicates the authorization level for the specified user (*UserName*) on the specified server (*ServerName*).

Class Settings

None.

Sample Code

```
Private Sub cmdTSGetAuthorizationType_Click()
      Dim lRtn As Long
      Dim tlinkName As String

      tlinkName = “LUCENT#G3_SWITCH#CSTA#SERVER”
      ‘Get the authorization type
      lRtn = px.TSGetAuthorizationType(tlinkName)
End Sub
```

CHAPTER 7

OCX Link Control Events

This chapter contains information regarding the returned events related to the telephony link methods sent by PhoneX.

In This Chapter

TSServerAvailable

Syntax:	*TSServerAvailable(ByVal ServerName As String, ByVal RetCode As Long)*
Description:	Collects a list of the available Telephony Servers available to the client machine.
Response to method:	[*TSListServers*]

Parameters

ServerName	The name of the Telephony Server found.
RetCode	Currently unused.

Usage Notes

This event fires when the OCX control has found a valid and operational Telephony Server. When no more Telephony Servers have been found, <*ServerName*> is blank and RetCode=0

Class Settings

None.

Sample Code

```
Private Sub px_TSServerAvailable(ByVal ServerName As
String, ByVal RetCode As Long)

      ‘Store ServerName into a list for future retrieval
where ColServer is a collection class with one ‘single
string property

      ColServer.Add ServerName

End Sub
```

TSMonitorStationReturn

Syntax:	*TSMonitorStationReturn(ByVal clsDevice As DeviceClass)*
Description:	Fires once PhoneX has successfully monitored the station device.
Response to method:	[*TSMonitorStation*]

Parameters

clsDevice	A device class containing all the information about the device being monitored.

Usage Notes

Once the device has been successfully monitored, PhoneX will request further information about the device from the Definity switch/MultiVantage server. This information will be reported in *ClassDeviceModified* events or *ClassCallModified* events.

If the *StreamVersion* is 5 or above, PhoneX will request the Definity switch/MultiVantage server to supply the station name. This information is appended to the *SwitchName* parameter of the class. A *ClassDeviceModified* event will be fired to indicate this change.

PhoneX will also request current device status information from the Definity switch/MultiVantage server. This will allow a current profile of the station device to be determined. If the station device is currently in use, call classes will be created in PhoneX that represent the call, its status and the devices that are present on the call. This newly created class will be reported to the container application via *ClassCallModified*.

Class Settings

DeviceDN	The DN (extension number) of the Definity/MultiVantage device that has been monitored.
MonitorType	The type of monitoring performed on the specified station. For monitor type values, refer to the enumeration *enMonitorType* in the *PhoneX Enumerations Guide*.

Sample Code

```
Private Sub px_TSMonitorStationReturn(ByVal clsDevice As
DeviceClass)
      ‘Make a call with this device
      Dim cal As CallClass

      Set cal = px.ActiveCallClasses.Add()
      If Not cal Is Nothing Then
            cal.CallerDN = clsDevice.DeviceDN
            cal.CalledDN = “8572“
            px.CallDial(cal)
      End If
End Sub
```

TSMonitorSkillReturn

Syntax:	*TSMonitorSkillReturn(ByVal clsDevice As DeviceClass)*
Description:	Fires once PhoneX has successfully monitored the skill device.

Response to method:	[*TSMonitorSkill*]

Parameters

clsDevice	A device class containing all the information about the device being monitored.

Usage Notes

Once the device has been successfully monitored, PhoneX will request further information about the device from the Definity switch/MultiVantage server. This information will be reported in a *ClassDeviceModified* event.

If the *StreamVersion* is 5 or above, PhoneX will request the Definity switch/MultiVantage server supply the skill name. This information is appended to the *SwitchName* parameter of the class. A *ClassDeviceModified* event will be fired to indicate this change.

Class Settings

DeviceDN	The DN (extension number) of the Definity/MultiVantage device that has been monitored.
MonitorType	The type of monitoring performed on the specified skill. For monitor type values, refer to the enumeration *enMonitorType* in the *PhoneX Enumerations Guide*.

Sample Code

```
Private Sub px_TSMonitorSkillReturn(ByVal clsDevice As
DeviceClass)
      If clsDevice.MonitorType = _
            enMonitorType.CompleteMonitor Then
            SkillMonitored = True
      Else
            SkillMonitored = False
      End If
 End Sub
```

TSMonitorVDNReturn

Syntax:	*TSMonitorVDNReturn(ByVal clsDevice As DeviceClass)*
Description:	Fires once PhoneX has successfully monitored the VDN.
Response to method:	[*TSMonitorVDN*]

Parameters

clsDevice	A device class containing all the information about the device being monitored.

Usage Notes

Once the device has been successfully monitored, PhoneX will request further information about the device from the Definity switch/MultiVantage server. This information will be reported in a *ClassDeviceModified* event.

If the *StreamVersion* is 5 or above, PhoneX will request the Definity switch/MultiVantage server to supply the VDN name. This information is appended to the *SwitchName* parameter of the class. A *ClassDeviceModified* event will be fired to indicate this change.

Class Settings

DeviceDN	The DN (extension number) of the Definity/MultiVantage device that has been monitored.
MonitorType	The type of monitoring performed on the specified VDN. For monitor type values, refer to the enumeration *enMonitorType* in the *PhoneX Enumerations Guide*.

Sample Code

See example for *TSMonitorSkillReturn*.

TSMonitorStopped

Syntax:	*TSMonitorStopped (ByVal clsDevice As DeviceClass, long MonitorEndCause)*
Description:	Fires when the Telephony Server stops monitoring a specific device.
Response to method:	None.

Parameters

clsDevice	A device class containing all the information about the device for which monitoring has been stopped.
MonitorEndCause	The cause value received from the Telephony Server that indicates why the monitoring has been stopped.

Usage Notes

The Telephony Server or the Definity switch/MultiVantage server may cancel monitoring for a device for a number of reasons including administrative changes on the Definity switch/MultiVantage server or a change in link status between the Definity switch/MultiVantage server and the Telephony Server. When this happens, the Telephony Server informs PhoneX which will fire this event.

Class Settings

```
Private Sub px_TSMonitorStopped(ByVal clsDevice As
DeviceClass, long MonitorEndCause)

  MsgBox “Monitoring has been stopped for device “ &
clsDevice.DeviceDN

End Sub
```

TSAuthorizationType

Syntax:	*TSAuthorizationType(ByVal ServerName As String, ByVal UserName As String, ByVal AuthType As Long)*
Description:	Indicates the authorization level for the specified user (*UserName*) on the specified server (*ServerName*).
Response to method:	[*TSGetAuthorizationType*]

Parameters

ServerName	The specified server name link is returned.
UserName	The specified user name the authorization is for.
AuthType	The authorization type for this user name. For authorization type values, refer to the enumeration *enAuthType* in the *PhoneX Enumerations Guide*.

Class Settings

None.

Sample Code

```
Private Sub px_TSAuthorizationType(ByVal ServerName As
String, ByVal UserName As String, ByVal AuthType As Long)

      If AuthType = enAuthType.atAuthLoginIDOnly Then

            ‘Start logging in

            px.PhoneXEnable = True

      End If

End Sub
```

TSLoggedIn

Syntax:	*TSLoggedIn(ByVal ActiveServer As enActiveServer, ByVal TServerVersion As String, ByVal StreamVersion As String, ByVal LoginReason As enLoginReason)*
Description:	Indicates the successful login attempt by PhoneX.

Response to:	The user setting the *PhoneXEnabled* property to True.

Parameters

ActiveServer	The Telephony Server link currently being used. This could be a primary, secondary or no link. For link type values, refer to the enumeration *enActiveServer* in the *PhoneX Enumerations Guide*.
TserverVersion	Telephony Server software version information.
StreamVersion	The private data stream version supported by the switch.
LoginReason	The reason the login was performed. This could be due to an error recovery login or a login performed by the user. For login reason values, refer to the enumeration *enLoginReason* in the *PhoneX Enumerations Guide*.

Class Settings

None.

Sample Code

```
Private Sub px_TSLoggedIn(ByVal ActiveServer As
enActiveServer, _ ByVal TServerVersion As String, ByVal
StreamVersion As String, _ ByVal LoginReason As
enLoginReason)

      DoStatus “TS Logged In Successfully!”

End Sub
```

TSLoggedOut

Syntax:	
Description:	Indicates ..
Response to:	

Parameters

Class Settings

Sample Code

CHAPTER 8

Call Control Methods

This chapter provides information on the use of methods for call control on a particular monitored station.

In This Chapter

CallAnswer

Syntax:	*CallAnswer(ByVal clsCall As CallClass) As Long*
Description:	Attempts to answer the call alerting at the monitored extension.
Returns:	Long

Parameters

clsCall	The call class to be answered.

Return Values

PxStreamFailed	The link to the Telephony Server has failed.
PxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
PxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
PxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *DeviceDN* exceeds the maximum number of character (64) or is zero length.
PxInvalidCallState	The call class is valid and contains valid parameter information but the call class is not in the alerting state.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Usage Notes

This will attempt to answer a call that is alerting on the current device. If there are any calls that are currently active, PhoneX will automatically hold these calls before attempting to answer the alerting call.

Return Events (in order fired)

CallAnswered()	PhoneX has successfully answered the call.

Error Event Values

Generic_State_Incompatibility

PhoneX is controlling an analogue telephone and the user failed to go off hook within the 5 second time period from the issue of the *CallAnswer*.

No_Call_To_Answer

The Definity ECS has redirected the call to coverage before the *CallAnswer* request was received.

Generic_System_Resource_Availability

This is an attempt to add a seventh party to a call with 6 active parties.

Resource_Busy

The PhoneX object already has an active call in the connected state that could not be held.

Sample Code

```
Private Sub cmdCallAnswer_Click()
      If cal Is Nothing Then
            Exit Sub
      Else
            px.CallAnswer cal
      End if
End Sub
```

CallConference

Syntax:	*CallConference(ByVal clsCall As CallClass, ByVal AddDN As String, ByVal ConfType As Long, ByVal UUI As String) As Long*
Description:	Creates a conference call by adding another party, *AddDN*, to the active call specified by the call class.
Returns:	Long

Parameters

clsCall	The existing call class that will have a new party added to it. This call must be in either the *csConnect* or *csHold* state.
AddDN	Destination that needs to be added to the current call. Additional special characters will be accepted in the *AddDN* field as defined in the appendix, Special Dial Characters.
ConfType	The type of conference to be undertaken. For conference type values, refer to the enumeration *enConferenceType* in the *PhoneX Enumerations Guide*.
UUI	Any user-to-user information that should be sent with the call to the *AddDN* destination.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.

pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *DeviceDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8).
pxInvalidCallState	The call class is valid and contains valid parameter information, however, the call class is not in the correct state. The existing call class must be in either the *csConnect* or *csHold* state. Additionally, if the device is in the process of performing a screened call conference and the CallConference method is called without the *ConfType* being *cfComplete* or *cfAbort*, this error will be returned.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Usage Notes

The call conference method allows an additional party to be added to an existing call. The call having the new party added must be in the connected (talking) state or on hold. This method can not be invoked for an alerting call. If the call is in the connected state, the call conference method will firstly hold the call before invoking a new call to the *AddDN* destination. If the call is already on hold, a new call is immediately generated to the *AddDN* destination.

To conference a call class that is on hold at the device and a call class that is in the connected state, use *CallJoin*.

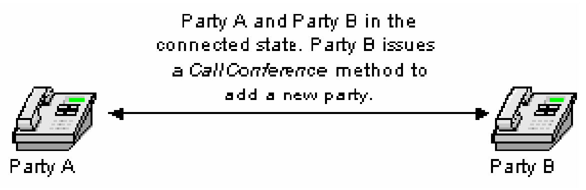
Conference Types

ctBlind	The conference is set up in an unscreened (blind) fashion. The initial call is placed on hold and a new call is placed to the destination specified in *AddDN*. Once the new call is in progress, the conference is completed.
ctWait	The current call is placed on hold and a new call is generated to the *AddDN* destination. Once this destination answers the call, the conference is completed.
ctScreened	The screened conference method is set up in two stages. Firstly, call the *CallConference* method specifying the *AddDN* and the *ctScreened* type. The initial call will be placed on hold and a new call placed to the *AddDN* parameter. To complete the conference, call the *CallConference* method specifying the *ctComplete* type.
ctComplete	The Complete conference type is used to join the two parties setup as a result of the *CallConference* with a *ctScreened* type.

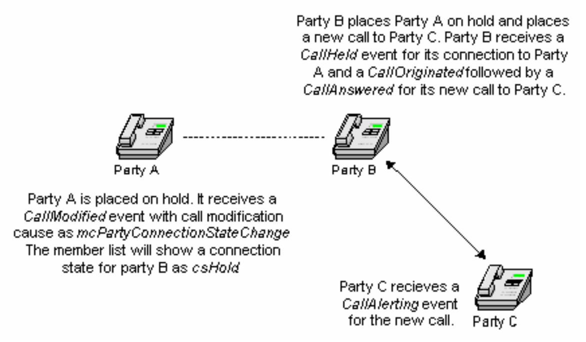
ctAbort	The conference abort is used to clear down the new party generated as a result of the *CallConference* with a *ctScreened* type. The original call remains in the held state.
Interactions	When multiple PhoneX users are involved as various parties to a conference call, the events fired differ depending on the activity the party is performing. When the conference is completed, all parties receive *CallModified* methods for the call class. For each party, however the reason for the modification is different. In general terms, the party that instigates the *CallConference* methods receives the *CallReleased* and *CallModified* events. Those parties who are passive participants to the conference call will receive *CallModified* events, with the event cause indicating that a new party has been added to the conference call.

Example

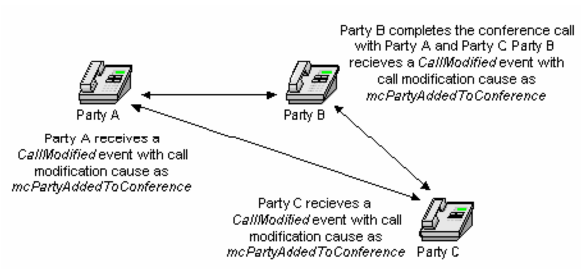
Step 1: Start Conference


Party A and Party B in the connected state. Party B issues a *CallConference* method to add a new party.
Party A
Party B

Step 2: Add New Party


Party B places Party A on hold and places a new call to Party C. Party B receives a *CallHeld* event for its connection to Party A and a *CallOriginated* followed by a *CallAnswered* for its new call to Party C.
Party A
Party B
Party A is placed on hold. It receives a *CallModified* event with call modification cause as *mcPartyConnectionStateChange* The member list will show a connection state for party B as *csHold*
Party C recieves a *CallAlerting* event for the new call.
Party C

Step 3: Complete Conference Call


Party B completes the conference call with Party A and Party C Party B recieves a *CallModified* event with call modification cause as *mcPartyAddedToConference*
Party A
Party B
Party A receives a *CallModified* event with call modification cause as *mcPartyAddedToConference*
Party C recieves a *CallModified* event with call modification cause as *mcPartyAddedToConference*
Party C

Return Events

CallHeld	The call specified by the call class has been held.
CallActive	A call has been successfully originated from the specified device.
CallOriginated	This event fires when PhoneX has completed making a call and the switch has decided to attempt the call.

CallDelivered	This event fires for an outbound call when the call has been presented to the destination device.
CallReleased	This event returns to the party that instigated the conference stating that the original call has been released.
CallModified	This event returns for all other parties that are in the conference call, including the conference controller. The reasons are, however, different between the passive participants and the controller.

Error Event Values

Conference_Member_Limit_Exceeded

The request attempted to add a seventh party to an existing six party conference call.

Sample Code

```
Private Sub cmdConference_Click()
      Dim cal As CallClass


      Set cal =
px.ActiveCallClasses.ItemActiveCall(“8575”)
      If Not cal Is Nothing And txtCall <> "" Then
            px.CallConference cal, txtCall, 1
      End If
End Sub
```

CallDial

Syntax:	*CallDial(ByVal clsCall As CallClass) As Long*
Description:	Places an outgoing call from *CallerDN* to the *CalledDN*. Optionally sends UUI to called party.
Returns:	Long

Parameters

clsCall	A call class containing the information necessary to make the call.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.

pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8).
pxInvalidCallState	The call class is valid and contains valid parameter information however, the call class is not in the correct state. A call dial method that is called where the call class *CallState* parameter is not *csIdle* will return this error.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Usage Notes

The *CallDial* service originates a call between two devices. The originator (*CallerDN*) must be a device on the switch.

If the call class does not exist within the PhoneX class collection, PhoneX will add it and make it available. If the *CallDial* fails (eg. invalid device/destination), the error will be reported through the *CallFailed* event (including the call class) or *TSError* mechanism (including the *invokeID*).

Class Settings

CallerDN	The DN that the call should be made from. This device must exist on the local Definity switch/MultiVantage server. The *CallerDN* parameter must be greater than 0 characters and less than 64 characters for the call class to attempt to originate a call. Any alpha characters (eg. a..z) will be converted to the numeric equivalent. Characters outside 0..9 or a..z will result in the method being rejected.
CalledDN	The destination that the call will be made to. The *CallerDN* parameter must be greater than 0 characters and less than 64 characters for the call class to attempt to originate a call. Any alpha characters (a..z) will be converted to the numeric equivalent. Additional special characters will be accepted in the *CalledDN* field as defined in the appendix, Special Dial Characters. If the destination is off net, the *CalledDN* parameter must contain any trunk access codes or ARS / AAR information necessary to allow the call to progress.
UUI	Any user-to-user information that may be required to be sent with the new call.

Return Events

CallActive	A call has been successfully originated from the specified device.
CallOriginated	This event fires when PhoneX has completed making a call and the switch has decided to attempt the call.
CallDelivered	This event fires for an outbound call when the call has been presented to the destination device.

Error Event Values

Invalid_Calling_Device

The device specified in the *CallingDN* is out of service or incorrectly administered on the switch.

Invalid_CSTA_Device_Identifier

The device specified in the *CallingDN* is invalid.

Generic_State_Incompatability

The originator does not go off hook and can not be forced off hook.

If *CallingDN* is an analog station, this must be taken off hook within five seconds of the *CallDial* being issued.

Sample Code

```
Private Sub Dial()
      Set cal = px.ActiveCallClasses.Add()

      If Not cal Is Nothing And txtCall <> "" Then
            cal.CallerDN = txtCaller
            cal.CalledDN = txtCalled
            cal.UUI = txtMsg
            px.CallDial cal
      End If
End Sub
```

CallDialDirectAgent

Syntax:	*CallDialDirectAgent(ByVal clsCall As CallClass, ByVal SplitSkill As String) As Long*
Description:	Places an outgoing call from the DN to the *AgentDN*.
Returns:	Long

Parameters

clsCall	The call class containing information to complete the call.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. Eg, the *CallerDN* or *CalledDN* exceeds the maximum number of characters (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior Release 8).
pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. A *CallDialDirectAgent* method that is called where the call class *CallState* parameter is not *csNone* or *csIdle* will return this error.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Usage Notes

This method will make a call to an agent DN directly. It attempts to originate a call between the DN and the agent logged into the specified split/skill. *CallDialDirectAgent* will then attempt to create a new call and establish a connection with the originating device first. This method is used mainly when the controlling application decides that the originator of the call needs to speak with a specific agent.

Class Settings

CallerDN	The DN the call should be made from. This device must exist on the local Definity switch/MultiVantage server. If the *CalledDN* contains a valid logical agent ID and the agent is logged in, the call will be made. However, if the agent is logged out, the direct agent call will be denied. The *CallerDN* parameter must be greater than 0 characters and less than 64 characters for the call class to attempt to originate a call. Any alpha characters (a..z, A..Z) will be converted to the numeric equivalent. Characters outside 0..9 or a..z, A..Z will result in the method being rejected.
CalledDN	The *CalledDN* must be a valid ACD agent extension. The *CalledDN* agent must be logged in to receive this call. If the agent is logged out, the direct agent call will be denied.
OtherDN	The *OtherDN* must contain a valid split extension. The agent specified in *CalledDN* must be a member of this split and must be logged in.

Return Events

CallActive	A call has been successfully originated from the specified device.
CallOriginated	Fires when PhoneX has completed making a call and the switch has decided to attempt the call.
CallDelivered	Fires for an outbound call when the call has been presented to the destination device.

Error Event Values

Invalid_Calling_Device

The device specified in the *CallingDN* is out of service or incorrectly administered on the switch.

Invalid_CSTA_Device_Identifier

The device specified in the *CallingDN* is invalid.

Generic_State_Incompatability

The originator does not go off hook and can not be forced off hook.

If *CallingDN* is an analog station, this must be taken off hook within 5 seconds of the *CallDialDirectAgent* being issued.

Sample Code

```
Private Sub cmdCallDialDirectAgent_Click()
      Dim lRtn As Long
      lRtn = px.CallDialDirectAgent(cal, txtSplitSkill)
End Sub
```

CallDialSupervisorAssist

Syntax:	*CallDialSupervisorAssist(ByVal clsCall As CallClass, ByVal SplitSkill As String) As Long*
Description:	Requests the specified DN to originate an outgoing call to a supervisor of a split/skill. Used if the application requires making a call to consult with another extension (typically the supervisor) within the ACD group.
Returns:	Long

Parameters

clsCall	Call class containing information to complete the call.
SplitSkill	The split/skill device the agent requires supervisor assistance for.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.

pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8).
pxInvalidState	The call class is valid and contains valid parameter information however, the call class is not in the correct state. A *CallDialSupervisorAssist* method that is called where the call class *CallState* parameter is not *csNone* or *csIdle* will return this error.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

CallActive	A call has been successfully originated from the specified device.
CallOriginated	This event fires when PhoneX has completed making a call and the switch has decided to attempt the call.
CallDelivered	This event fires for an outbound call when the call has been presented to the destination device.

Error Event Values

Invalid_Calling_Device

The device specified in the *CallingDN* is out of service or incorrectly administered on the switch.

Invalid_CSTA_Device_Identifier

The device specified in the *CallingDN* is invalid.

Generic_State_Incompatability

The originator does not go off hook and can not be forced off hook.

If *CallingDN* is an analog station, this must be taken off hook within 5 seconds of the *CallDialSupervisorAssist* being issued.

Sample Code

```
Private Sub cmdCallDialSupervisorAssist_Click()

      Dim lRtn As Long

      lRtn =
px.CallDialSupervisorAssist(cal,splitskillTxt)

End Sub
```

CallDivert

Syntax:	*CallDivert(ByVal clsCall As CallClass, ByVal DivertDN As String) As Long*
Description:	Diverts (deflects) the alerting incoming call to the *DivertDN*.
Returns:	Long

Parameters

clsCall	The *CallClass* object of the call alerting at the device that is to be diverted.
DivertDN	Destination to divert the call to.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8). Or the *DivertDN* parameter supplied contains invalid characters.
pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. A *CallDivert* method that is called where the call class *CallState* parameter is not *csNone* or *csIdle* will return this error.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

CallDiverted	The alerting call has successfully been diverted to a new destination as specified.

Error Event Values

Invalid_Calling_Device

The device specified in the *DivertDN* is out of service or incorrectly administered on the switch.

Sample Code

In the user interface of the sample application there is the facility to turn on call diverting and specify an extension to divert the call to. If it is turned on and there is a number to divert the call to then it is diverted before it can be answered.

```
Private Sub px_CallAlerting(ByVal clsCall As _
      Object, ByVal EventCause As Long)
      ‘This event is triggered when a call is received on
the monitored device. Save the call identifier ‘of the
incoming call. This identifier is used outside of this
event to retrieve this call
      curCall = clsCall.CallIdentifier
      If optOn And txtDivertTo <> "" Then
            px.CallDivert clsCall, txtDivertTo
      Else ‘Call divert is not activated
            cmdAnswer.Enabled = True
            cmdHangup.Enabled = True
            cmdMakeCall.Enabled = False
            DoStatus "Incoming call"
      End If
End Sub
```

CallHold

Syntax:	*CallHold(ByVal clsCall As CallClass) As Long*
Description:	Places the active call on the specified DN on hold.
Returns:	Long

Parameters

clsCall	The call class object that is to be held.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	One of the call classes passed with this method is not known to PhoneX and can not be added to the list of active calls.
pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. A *CallHold* method that is called where the call class *CallState* parameter is not *csConnect* or *csActive* will return this error.

invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

CallHeld	The call specified by the call class has been held

Error Event Values

Invalid_Calling_Device

The device specified in the *CallingDN* is out of service or incorrectly administered on the switch.

Sample Code

This example gets the current active call and checks the state of it. The current call is placed on hold if it is in the connected state or it is placed in the connected state if it is on hold. Otherwise it remains unaffected.

```
Private Sub cmdHold_Click()

‘curCall is retrieved from the CallAlerting, CallAnswered
and CallOriginating events

      Dim cal As CallClass

      Set cal = px.ActiveCallClasses.Item(curCall)

      If Not cal Is Nothing Then

            If cal.CallState = 4 Then

                  ‘The call is on hold

                  px.CallUnhold cal

            ElseIf cal.CallState = 3 Then

                  ‘If there is a connected call

                  px.CallHold cal

            End If

      End If

End Sub
```

CallJoin

Syntax:	*CallJoin(ByVal clsCallA As CallClass, ByVal clsCallB As CallClass, ByVal JoinType As Long) As Long*
Description:	Joins two calls that exist on a single station device. The call join will allow two calls to be transferred together, in which case the new call leaves the primary device, or to be conferenced together to create a multi-party conference.
Returns:	Long

Parameters

clsCallA	The first call class to be joined.
clsCallB	The second call class to be joined.
JoinType	Determines whether the calls will be transferred together or conferenced together.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	One of the call classes passed with this method does not point to a valid class object.
pxInvalidClass	One of the supplied call classes is not known and can not be included in the list of call classes.
pxInvalidCallState	One of the supplied call classes is valid and contains valid parameter information however the call class is not in the correct state. The *CallJoin* method requires one call class specified to be in a held state and one to be in an active state.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

CallReleased	This event shall return to the caller that instigated the *CallJoin* method.
CallModified	Joining the calls was successful.

Error Event Values

Invalid_Calling_Device

The device specified in the *CallingDN* is out of service or incorrectly administered on the switch.

Sample Code

```
Private Sub cmdCallJoin_Click()
      Dim lRtn As Long


      ‘Originally, we have two call classes. CalA and CalB
      If CalA.CallState = CS_HELD Then
            lRtn = px.CallJoin(CalA,CalB,1)
      Else
            lRtn = px.CallJoin(CalB,CalA,2)
      End If
End Sub
```

CallListenHold

Syntax:	*CallListenHold(ByVal clsCall As CallClass, ByVal selectedParty As String, ByVal AllPartyHold As Boolean) As Long*
Description:	Disconnects a call party's listen hold path from the active conversation. This effectively excludes that party from hearing any further part of the conversation. The talk path for the party is intact.
Returns:	Long

Parameters

clsCall	The call class representing the call in progress.
SelectedParty	The selected party that is to be listen held from the party specified in the call class.
AllPartyHold	If True, the listening paths of all parties on the call will be held from the device in the call class. The *SelectedParty* parameter is ignored. If False, only the *SelectedParty* parameter is held from the device in the call class.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8). Or the *selectedParty* parameter supplied contains invalid characters.
pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. A *CallListenHold* method that is called where the call class *CallState* parameter is not *csConnect* will return this error.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

CallListenHeld	This event indicates that the request has been successful. The call class will contain the current status of the connection state of all parties to the call.

Error Event Values

Invalid_Calling_Device

The device specified in the *CallingDN* is out of service or incorrectly administered on the switch.

Value_Out_Of_Range

A party specified is not part of the call or is in the wrong state. The selected party can not be in the alerting state.

Invalid_Object_State

The request to listen hold from all parties has not been granted because there are no other eligible parties on the call.

Generic_System_Resource_Availability

Switch capacity has been exceeded.

Sample Code

```
Private Sub cmdListenHold_Click()
      Dim lRtn As Long
      If cal Is Nothing Then
            Exit Sub
      Else
            lRtn =
px.CallListenHold(cal,partyTxt,False)
      End if
End Sub
```

CallListenUnHold

Syntax:	*CallListenUnHold(ByVal clsCall As CallClass,ByVal selectedParty As String, ByVal AllPartyHold As Boolean) As Long*
Description:	Retrieves party from selective listen hold. Reverses the *CallListenHold* status for a specific device on the current call.
Returns:	Long

Parameters

clsCall	The call class that specifies the subject device to retrieve from *CallListenHold*.
SelectedParty	The party whose listen path is to be re-connected to the device specified in the call class.

AllPartyHold	If set to True, the listening paths of all parties on the call will be reconnected to the device specified in the call class and the *SelectedParty* parameter is ignored. If False only the party specified in the *SelectedParty* parameter is connected.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8). Or the *selectedParty* parameter supplied contains invalid characters.
pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. A *CallListenUnHold* method that is called where the call class *CallState* parameter is not *csHold* will return this error.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

CallListenUnHeld	The *CallListenUnHeld* indicates that the request has been successful.

Error Event Values

Invalid_Calling_Device

The device specified in the *CallingDN* is out of service or incorrectly administered on the switch.

Value_Out_Of_Range

A party specified is not part of the call or is in the wrong state. The selected party can not be in the alerting state.

Invalid_Object_State

The request to listen hold from all parties has not been granted because there are no other eligible parties on the call.

Generic_System_Resource_Availability

Switch capacity has been exceeded.

Sample Code

```
Private Sub cmdListenUnHold_Click()
      Dim lRtn As Long


      If cal Is Nothing Then
            Exit Sub
      Else
            lRtn =
px.CallListenUnHold(cal,partyTxt,True)
      End if
End Sub
```

CallPartyDrop

Syntax:	*CallPartyDrop(ByVal clsCall As CallClass, ByVal DropDN As String) As Long*
Description:	Drops a specific party from the existing call.
Returns:	Long

Parameters

clsCall	The call class that the party should be removed from
DropDN	The party that should be removed from the call.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8). Or the *DropDN* parameter supplied contains invalid characters.
pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. A *CallPartyDrop* can not be called for a call class that is alerting at the specified device.

invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Usage Notes

Where the call is a two-party call, this results in the call being cleared. For a multi party conference, any party can be removed from the call, regardless of the order in which the parties were added.

In a multi-party conference, using the *CallPartyDrop* method does not clear down the entire call even when the dropped party is the only "on switch" device.

A conference call with one station and two trunk parties in progress, issuing a *CallPartyDrop* for the station device will leave the two trunk parties connected.

Refer *CallRelease*.

Return Events

CallPartyDropped	A party has been removed from the call.

Error Event Values

Value_Out_Of_Range

A party specified is not part of the call or is in the wrong state. The selected party can not be in the alerting state.

Generic_System_Resource_Availability

Switch capacity has been exceeded

Sample Code

```
Private Sub cmdHangup_Click()
      'Hang up currently active call
      'curCall is retrieved from the
CallAlerting,CallAnswered and CallOriginating events
      Dim cal As CallClass

      'curCall1 = call identifier of the current call
      Set cal = px.ActiveCallClasses.Item(curCall1)
      If Not cal Is Nothing Then
            px.CallRelease cal
      End If
End Sub
```

CallRelease

Syntax:	*CallRelease(ByVal clsCall As CallClass) As Long*

Description:	Releases all connections on the specified call appearance (the default is the active call appearance), including all members of an established conference call. The call class is returned to the idle state and moved to the old call list.
Returns:	Long

Parameters

clsCall	The call class representing the call to be released.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8).
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

CallPartyDropped	There may be more than one *CallPartyDropped* event that occurs depending on the number of parties this method is invoked for.

Error Event Values

Value_Out_Of_Range

A party specified is not part of the call or is in the wrong state. The selected party can not be in the alerting state.

Generic_System_Resource_Availability

Switch capacity has been exceeded

Sample Code

```
Private Sub cmdHangup_Click()
      ‘Hang up currently active call
      Dim cal As CallClass
      ‘curCall is the call identifier of the current call
      Set cal = px.ActiveCallClasses.Item(curCall)
      If Not cal Is Nothing Then
            px.CallRelease cal
      End If
End Sub
```

CallSendDTMF

Syntax:	*CallSendDTMF(ByVal clsCall As CallClass, ByVal DTMFDigits As String) As Long*
Description:	Generates DTMF digits on the specified monitored DN and sends them to all parties on the call. The call must be in the connected state.
Returns:	Long

Parameters

clsCall	The call class representing the active call that requires the DTMF string to be outpulsed.
DTMFDigits	A string of dialable characters that will be out pulsed as in-band DTMF. This string must be less than 32 characters and only contain the characters 0..9, a..z, A..Z, *, #.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	The *DTMFDigits* parameter contains invalid information. The parameter must have a length greater than 0 and less than 32 characters and only contain characters 0..9, a..z, A..Z, *, #.
pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

None.

Error Event Values

Value_Out_Of_Range

A party specified is not part of the call or is in the wrong state. The selected party can not be in the alerting state.

Generic_System_Resource_Availability

Switch capacity has been exceeded

Sample Code

When a call is in the connected state, the sample application provides the facility to send DTMF signals using an interface that resembles the keypad on a phone. Here is the code from the click event for the hash button:

```
Private Sub cmdHash_Click()

      ‘curCall is retrieved from the CallAlerting,
CallAnswered and CallOriginating events

      Dim cal As CallClass

      Set cal = px.ActiveCallClasses.Item(curCall)

      If Not cal Is Nothing Then

            Px.CallSendDTMF cal, cmdHash.Caption

      End If

End Sub
```

CallTransfer

Syntax:	*CallTransfer(ByVal clsCall As CallClass, ByVal DestinationDN As String, ByVal TransferType As Long, ByVal UUI As String, ByVal Persist As Boolean) As Long*
Description:	Transfers the active call on the specified monitored DN to another DN, optionally specifying the transfer method (default=BLIND)
Returns:	Long

Parameters

clsCall	The call class representing the active call to be transferred.
DestinationDN	Destination that the active call will be transferred to. Additional special characters will be accepted in the *DestinationDN* field as defined in the appendix, Special Dial Characters.
TransferType	The type of transfer to be undertaken. For transfer type values, refer to the enumeration *enTransferType* in the *PhoneX Enumerations Guide.*
UUI	Any user-to-user information to be sent with the call.
Persist	Reserved. Not used.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8). Or the *DestinationDN* parameter supplied contains invalid characters.
pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. The call represented by the call class must be active at the monitored device. A call alerting at the specified device can not be transferred.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Usage Notes

The *CallTransfer* method allows an existing call to be transferred to a new destination. The call being transferred in the connected (talking) state or be on hold. This method can not be invoked for a call alerting at the monitored station. If the call is in the connected state, the call transfer method will firstly hold the call before invoking a new call to the *DestinationDN* destination. If the call is already on hold, a new call is immediately generated to the *DestinationDN* destination.

To transfer a call class that is on hold at the device to a call class that is in the connected state, use *CallJoin*.

Transfer Types

ttBlind	The transfer is set up in an unscreened (blind) fashion. The initial call is placed on hold and a new call is placed to the destination specified in *DestinationDN*. Once the new call is in progress, the transfer is completed.
ttWait	The current call is placed on hold and a new call is generated to the *DestinationDN* destination. Once this destination answers the call, the transfer is completed.
ttScreened	The screened transfer is set up in a two-stage method. Firstly, call the *CallTransfer* method specifying the *DestinationDN* and the *ttScreened* type. The initial call will be placed on hold and a new call placed to the *DestinationDN* parameter. To complete the transfer, call the *CallTransfer* method specifying the *ttComplete* type.

ttComplete	The complete transfer type is used to join the two parties set up as a result of the *CallTransfer* with a *ttScreened* type.
ttAbort	The transfer abort is used to clear down the new party generated as a result of the *CallTransfer* with a *ttScreened* type. The original call remains in the held state.
Interactions	When multiple PhoneX users are involved as various parties to a conference call, the events fired differ depending on the activity the party is performing. When the conference is completed, all parties receive *CallModified* events for the call class. For each party, however, the reason for the modification is different. In general terms, the party that instigates the *CallTransfer* methods receives a *CallTransferred* event, those parties who are passive participants to the conference call will receive *CallModified* events with the event cause indicating that a new party has been added to the conference call.

Return Events

CallHeld	The call specified by the call class has been held.
CallActive	A call has been successfully originated from the specified device.
CallOriginated	Fires when PhoneX has completed making a call and the switch has decided to attempt the call.
CallDelivered	Fires for an outbound call when the call has been presented to the destination device.
CallModified	Returns for all the passive parties in the transfer.
CallReleased	Returns to the caller that instigated the transfer.

Error Event Values

Value_Out_Of_Range

A party specified is not part of the call or is in the wrong state. The selected party can not be in the alerting state.

Generic_System_Resource_Availability

Switch capacity has been exceeded

Sample Code

If there is a connected call then the transfer button is enabled. The button caption will change when the call is being transferred. The action performed by this code is determined by the button caption.

This example illustrates a blind transfer and how to cancel a transfer.

```
Private Sub Transfer()

      ‘curCall is retrieved from the
CallAlerting,CallAnswered and CallOriginating events

      Dim cal As CallClass
      Set cal = px.ActiveCallClasses.Item(curCall)
      If Not cal Is Nothing And txtTransferTo <> "" _
            And cmdTransfer.Caption = transText Then
            'Transfer the call
            cmdTransfer.Caption = transCancelText
            px.CallTransfer cal, txtTransferTo, 1
      ElseIf cmdTransfer.Caption = TransCancelText Then
            'Call is currently being transferred,so
cancel the Transfer
            cmdTransfer.Caption = transText
            If Not cal Is Nothing Then
                  px.CallTransfer cal, "", 5
            End If
      End If
End Sub
```

CallUnHold

Syntax:	*CallUnHold(ByVal clsCall As CallClass) As Long*
Description:	Returns the held call on the specified DN call appearance to the active state.
Returns:	Long

Parameters

clsCall	The call class that is required.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length, or the UUI exceeds 96 characters (96 if you have a Release 8 Definity ECS with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server, or 32 if your switch is prior to Release 8).

pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. A *CallUnHold* method that is called where the call class *CallState* parameter is not *csHold* will return this error.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

CallUnHeld	Returns if the call to the *CallUnHold* method was successful.

Error Event Values

Value_Out_Of_Range

A party specified is not part of the call or is in the wrong state. The selected party can not be in the alerting state.

Generic_System_Resource_Availability

Switch capacity has been exceeded

Sample Code

See example for *CallHold*.

DeviceMute

Syntax:	*DeviceMute(ByVal clsCall As CallClass) As Long*
Description:	Mutes the active call on the specified DN.
Returns:	Long

Parameters

clsCall	The call class that is to be muted.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.
pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of characters (64) or is zero.

pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. A *DeviceMute* method that is called where the call class *CallState* parameter is not *csActive* will return this error.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

DeviceMuted	Fires if the attempt to mute the call was successful.

Error Event Values

Value_Out_Of_Range

A party specified is not part of the call or is in the wrong state. The selected party can not be in the alerting state.

Generic_System_Resource_Availability

Switch capacity has been exceeded

Sample Code

```
Private Sub cmdDeviceMute_Click()

      If Not cal Is Nothing Then

            px.DeviceMute cal

      End If

End Sub
```

DeviceUnMute

Syntax:	*DeviceUnMute(ByVal clsCall As CallClass) As Long*
Description:	Un-mutes the active call on the specified DN.
Returns:	Long

Parameters

clsCall	The call class that is to be unmuted.

Return Values

pxStreamFailed	The link to the Telephony Server has failed.
pxClassEmpty	The *clsCall* passed with this method does not point to a valid class object.
pxInvalidClass	The class object, *clsCall*, is not known and can not be included in the list of active calls.

pxInvalidParameter	One (or more) of the class device parameters is invalid. For example, the *CallerDN* or *CalledDN* exceeds the maximum number of character (64) or is zero length.
pxInvalidCallState	The call class is valid and contains valid parameter information however the call class is not in the correct state. A *DeviceUnMute* method that is called where the call class *CallState* parameter is not *csActive* will return this error.
invokeID	Any number greater than 1000 that is returned from the method is considered an *invokeID*. This *invokeID* will be returned with the event confirmation or any error event that is generated.

Return Events

DeviceUnMuted	This event will fire if the attempt to unmute the call was successful.

Sample Code

See example for *DeviceMute*.

CHAPTER 9

Call Control Events

This chapter contains information regarding the events that fire when methods were called.

In This Chapter

CallActive

Syntax:	*CallActive(ByVal clsCall As CallClass)*
Description:	Fires when a call appearance becomes active (goes off-hook) and the user receives dial tone as part of an outbound call. This event is also received as part of the event sequence that results from a successful *CallDial* method.
Response to method:	None.

Parameters

clsCall	The *CallClass* object that has been created as a result of the device being taken off hook.

Class Settings

CallID	The *CallID* method returns the Definity ECS generated call ID for the new call.
CallIdentifier	The *CallIdentifier* method returns the identifier for the call that has been created. This identifier will be used when the controlling application wishes to retrieve the *CallClass* object via the *ActiveCallClasses.Item* method, the *ActiveCallClasses.ItemActiveCall* method or the *ActiveCallClasses.ItemCallByCallAppearance* method.
DN	The DN method returns the string property indicating the device that has been taken off hook.
UCID	The Definity ECS assigned UCID will be in the UCID property if the Definity ECS has been configured to supply it. As this is a new *CallClass* object, all other class properties will contain default information.

Sample Code

```
Private Sub px_CallActive(ByVal clsCall As CallClass)

      DoStatus "Off hook"

End Sub
```

CallAlerting

Syntax:	*CallAlerting(ByVal clsCall As CallClass, ByVal EventCause As Long)*
Description:	Fires when the monitored DN has received a call without being answered.
Response to method:	None.

Parameters

clsCall	*CallClass* object that has been created as a result of the call being delivered to the monitored device.
EventCause	Specifies the cause for this event. For event cause values, refer to the enumeration *enEventCause* in the *PhoneX Enumerations Guide*.

Class Settings

DN	The DN method returns the directory number that is alerting. This DN is the station being controlled by the PhoneX object.
CalledDN	The *CalledDN* property contains the number that was called by the incoming party.
CalledName	If PhoneX has information about the number that was dialed, eg. a monitored VDN, the name of that device will be included in the *CalledName* property in the *CallClass* object. This name is retrieved from the Definity switch/MultiVantage server when the controlling application starts using PhoneX.
CallerDN	The *CallerDN* property contains the available information about the caller number. If the call is internal, the number of the Definity switch/MultiVantage server subscriber is displayed. If the call is received from the public network and calling line information is received (CLI, ANI), this is displayed.
CallerName	If this information is available, the Definity/MultiVantage name for the calling party is displayed.
UUI	Any user to user information that is received with the call is contained within this class property.

Sample Code

```
Private Sub px_CallAlerting(ByVal clsCall As CallClass,
ByVal EventCause As Long)

      DoStatus "Call Alerting"

End Sub
```

CallAnswered

Syntax:	*CallAnswered(ByVal clsCall As CallClass, ByVal AnsweredDN As String, ByVal EventCause As Long)*
Description:	Fires when a call originated from a monitored DN is answered.
Response to method:	[*CallAnswer*]

Parameters

clsCall	The *CallClass* object that has been answered.
AnsweredDN	Specifies the device that joined the call.
EventCause	Specifies the cause for this event. For event cause values, refer to the enumeration *enEventCause* in the *PhoneX Enumerations Guide*.

Usage Notes

This event fires when the call represented by the *CallClass* object is answered. Information in the *CallClass* object remains unchanged from the *CallAlerting* event with the exception of the *CallState*.

Class Settings

None.

Sample Code

This example enables/disables the appropriate buttons on the user interface and updates the status bar.

```
Private Sub px_CallAnswered(ByVal clsCall As CallClass,
ByVal AnsweredDN As String)


      ‘This event is triggered when an incoming or
outgoing call is answered. Save the call identifier ‘of
the answered call. This identifier is used outside of this
event to retrieve this call


      curCall = clsCall.CallIdentifier

      ‘Enable and disable the appropriate functions on the
user interface

      cmdAnswer.Enabled = False

      cmdHold.Enabled = True

      cmdHold.Caption = holdText

      cmdHangup.Enabled = True

      cmdConference.Enabled = True

      fraTransfer.Enabled = True

      ‘Update status bar

      If clsCall.CallDirection = 0 Then

            DoStatus "Incoming call in progress..."

      Else

            DoStatus "Outgoing call in progress..."

      End If

End Sub
```

CallDelivered

Syntax:	*CallDelivered(ByVal clsCall As CallClass, ByVal EventCause As Long)*
Description:	Fires when an outgoing call has been delivered to the required destination.
Response to method:	[*CallDial, CallConference, CallTransfer, CallDialDirectAgent, CallDialSupervisorAssist*]

Parameters

clsCall	The *CallClass* object representing the outbound call.
EventCause	Specifies the cause for this event. For event cause values, refer to the enumeration *enEventCause* in the *PhoneX Enumerations Guide*.

Class Settings

None.

Sample Code

```
Private Sub px_CallDelivered(ByVal clsCall As _
CallClass, ByVal EventCause As Long)
      DoStatus "The destination has been reached "
End Sub
```

CallFailed

Syntax:	*CallFailed(ByVal clsCall As CallClass)*
Description:	Fires when an outbound call from the monitored device has failed.
Response:	[*CallDial*]

Parameters

clsCall	The *CallClass* object representing the call that has failed.

Usage Notes

The *clsCall* parameter will indicate the *CallClass* object that has failed.

This event will fire when the destination of a call is busy or unavailable, or a call receives reorder/denial treatment as described in the *Definity ECS Programmer’s Guide for CentreVu CTI.*

Class Settings

None.

Sample Code

```
Private Sub px_CallFailed(ByVal clsCall As CallClass)
      DoStatus "The call has failed"
End Sub
```

CallDiverted

Syntax:	*CallDiverted(ByVal clsCall As CallClass, ByVal NewDestination As String)*
Description:	Fires when an incoming call was diverted to a new destination successfully. This event fires for the deflected call case and for the Send All Calls feature.
Response:	[*CallDivert*]

Parameters

clsCall	The *CallClass* object representing the call that was diverted.
NewDestination	The new DN that got the diverted call.

Class Settings

None.

Sample Code

```
Private Sub px_CallDiverted(ByVal clsCall As CallClass,
ByVal _ NewDestination As String)
      DoStatus "The call has been diverted to" +
NewDestination
 End Sub
```

CallHeld

Syntax:	*CallHeld(ByVal clsCall As CallClass)*
Description:	Fires when a call has been placed on hold.
Response to method:	[*CallHold*]

Parameters

clsCall	The *CallClass* object representing the call that has been held.

Usage Notes

This event fires to indicate the call represented by the *CallClass* object is placed into the held state. This may be the result of the *CallHold* method being called or as a result of the user activating the hold feature from the feature phone.

Class Settings

None.

Sample Code

```
Private Sub px_CallHeld(ByVal clsCall As CallClass)
      cmdHold.Caption = unholdText
      DoStatus "The call is on hold"
End Sub
```

CallListenHeld

Syntax:	*CallListenHeld(ByVal clsCall As CallClass, ByVal selectedParty As String)*
Description:	Fires when the monitored device places a party on the current call on listen hold.
Response to method:	[*CallListenHold*]

Parameters

clsCall	The call class representing the call that was put on listen hold.
SelectedParty	The party on listen hold. The *SelectedParty* is not able to hear the other member(s) on the call but the other members can hear the *SelectedParty*.

Class Settings

None.

Sample Code

```
Private Sub px_CallListenHeld(ByVal clsCall As CallClass,
ByVal selectedParty As String)
      DoStatus "The call is on listen hold"
End Sub
```

CallListenUnHeld

Syntax:	*CallListenUnHeld(ByVal clsCall As CallClass, ByVal selectedParty As String)*
Description:	Fires when a call has been retrieved from listen hold.
Response to method:	[*CallListenUnHold*]

Parameters

clsCall	The call class representing the call that was retrieve from listen hold.
selectedParty	The party retrieved from listen hold, giving them the ability to hear the other member(s) on the call again.

Class Settings

None.

Sample Code

```
Private Sub px_CallListenUnHeld(ByVal clsCall As
CallClass, ByVal selectedParty As String)
      DoStatus "The call is on listen unhold"
End Sub
```

CallModified

Syntax:	*CallModified(ByVal clsCall As CallClass)*
Description:	Fires when a call has been modified.
Response to method:	[*CallConference, CallTransfer, CallDivert*]

Parameters

clsCall	The *CallClass* object that has been modified.

Class Settings

CallModifiedMethod	The reason the call was modified. For reason values, refer to the enumeration *enModifyCause* in the *PhoneX Enumerations Guide*.
Added_To_Conference	The device being monitored has been added to a conference.
Party_Added_To_Conference	Another party has been added to the conference.
Transferred_To	This DN has been transferred to another party.

Transferred_From	A call has been transferred to this DN, ie. another party has completed the transfer passing the original caller to this DN.
Diverted_To	A call from this DN to party B, has been diverted (*CallDivert*) at party B to another destination.
Party_Connection_State_Change	The connection state of another party on the call has changed. For example, during a call between this DN and party B, party B has placed their call appearance on hold. Connection states of all parties can be determined from the *MemberList* property in the *CallClass* object.
Party_Talk_State_Change	The talk state of another party on the call has changed. For example, during a call between this DN and party B, party B has placed their connection on listen hold. Connection states of all parties can be determined from the *MemberList* property in the *CallClass* object.

Sample Code

```
Private Sub px_CallModified(ByVal clsCall As CallClass)

      ‘Update the status bar with the reason the call was
modified

      Dim nMembers As Integer


      NMembers =
clsCall.MemberList.CurrentNumberOfMembers

      Select Case clsCall.CallModifiedMethod

      Case 1: ‘Add party to conference call

            DoStatus "You have been added to a " _

            + "conference call with " + Str(nMembers) +
" members"

      Case 2: ‘Count the total number of calls in
conference

            DoStatus Str(nMembers) + " calls in
conference"

            cmdDrop.Enabled = True

      Case 3: ‘Transfer call to another device

            DoStatus "Your call has been transferred To
" + clsCall.NewDN

      Case 4: ‘Transfer call from another device

            DoStatus "Incoming call (" + clsCall.CallerDN
+ _

            ") has been transferred from " +
clsCall.OtherDN
```

```
      Case 5: ‘Divert call from the called device
            DoStatus "Your call has been diverted to " +
clsCall.OtherDN
      Case 6:
            DoStatus "Your connection state has changed."
      Case 7:
            DoStatus "Your talk state has changed."
      End Select
End Sub
```

CallNetworkReached

Syntax:	*CallNetworkReached(ByVal clsCall As CallClass)*
Description:	Fires when a call reaches the telephone network.
Response to method:	[*CallDial*]

Parameters

clsCall	The *CallClass* object that has reached the network interface

Usage Notes

The *NetworkReached* event will fire to indicate one of the following events:

- A non-ISDN call has cut through the switch boundary to another network
- An ISDN call is leaving the ISDN network.

A *NetworkReached* event will never fire for a call to a device that is directly connected to the Definity switch/MultiVantage server.

Multiple *NetworkReached* events may be received for a single call.

Class Settings

None.

Sample Code

```
Private Sub px_CallNetworkReached(ByVal clsCall As
CallClass)
      DoStatus "Network Reached Event"
End Sub
```

CallOriginated

Syntax:	*CallOriginated(ByVal clsCall As CallClass)*

Description:	Fires when a call attempt has been originated.
Response to method:	[*CallDial*]

Parameters

clsCall	The *CallClass* object that has been originated.

Usage Notes

The *CallOriginated* event is generated as follows:

- When the station user completes dialing a valid number
- When the *CallDial* is invoked and the switch determines that a call is to be attempted.

The event will not be triggered when a call is aborted because an invalid number has been provided or because the device DN that PhoneX is controlling is not allowed to originate the call (via COR).

Class Settings

None.

Sample Code

```
Private Sub px_CallOriginated(ByVal clsCall As Object)

      ‘This event is triggered once an outgoing call is
placed. Save the call identifier of the answered ‘call.
This identifier is used outside of this event to retrieve
this call


      curCall = clsCall.CallIdentifier

      cmdHangup.Enabled = True

      cmdAnswer.Enabled = False

      cmdConference = False

      DoStatus "Outgoing call in progress"

End Sub
```

CallPartyDropped

Syntax:	*CallPartyDropped(ByVal clsCall As CallClass, ByVal DroppedDN As String)*
Description:	Fires when the requested party has been dropped from a call. This will occur if the *CallPartyDrop* method is called and the outcome is successful.
Response to method:	[*CallPartyDrop*]

Parameters

clsCall	The *CallClass* object that has had the party removed from it.

DroppedDN	The DN that has left the call.

Class Settings

None.

Sample Code

Different actions are performed depending on the current call state.

```
Private Sub px_CallPartyDropped(ByVal clsCall _
      As CallClass, ByVal DroppedDN As String)


      ‘This event is triggered when a call is terminated
or a party is dropped from a conference call
      Dim Nmembers As Integer
      Nmembers =
clsCall.MemberList.CurrentNumberOfMembers
      Select Case clsCall.CallState
      Case 0: ‘Enable and disable the appropriate
functions on the user interface for the Idle case
            cmdHold.Enabled = False
            cmdHangup.Enabled = False
            cmdAnswer.Enabled = False
            fraTransfer.Enabled = False
            cmdConference = False
            cmdMakeCall.Enabled = True
            DoStatus "Call terminated"
      Case 3: ‘Respond according to the number of devices
connected to the current call for the ‘connected case
            Select Case nMembers
            Case Is > 2: ‘Conference call in progress
                  DoStatus Str(nMembers) + " calls in
conference"
            Case 2: ‘Enable and disable the appropriate
functions on the user interface
                  cmdDrop.Enabled = False
                  cmdHangup.Enabled = True
                  If clsCall.CallDirection = 0 Then
                        DoStatus "Incoming call in
progress"
                  Else
                        DoStatus "Outgoing call in
progress"
                  End If
            End Select
      End Select
End Sub
```

CallPhoneActive

Syntax:	*CallPhoneActive(ByVal DN As String)*
Description:	Fires when the monitored DN is set to Busy Indicator (Basic) mode and the phone returns events of type *CallActive*, *CallAlerting* or *CallAnswered.*
Response to method:	[*CallActive, CallAlerting, CallAnswered*]

Parameters

DN	The station that is monitored as a busy indicator that received an *Active*, *Alerting* and/or *Answered* event.

Class Settings

None.

Sample Code

```
Private Sub px_CallPhoneActive(ByVal DN As String)
      DoStatus "Phone Is Off Hook"
End Sub
```

CallPhoneNotActive

Syntax:	*CallPhoneNotActive(ByVal DN As String)*
Description:	Fires when the monitored DN is set to Busy Indicator (Basic) mode and the phone returns events for *CallRelease* or *CallPartyDropped.*
Response to method:	[*CallReleased, CallPartyDropped*]

Parameters

DN	The station that is monitored as a busy indicator that received the *CallReleased* and/or *CallPartyDropped* event.

Class Settings

None.

Sample Code

```
Private Sub px_CallPhoneNotActive(ByVal DN As String)

      DoStatus "Phone Is On Hook"

End Sub
```

CallQueued

Syntax:	*CallQueued(clsCall As CallClass, queue As String, numberQueued As Long)*
Description:	Fires when a call has been queued to a split or skill.
Response to method:	[*CallDial*]

Parameters

clsCall	The call class representing the call that was put in the queue.
queue	Specifies the queuing device to which the call has queued. This is the extension of the ACD split to which the call queued.
numberQueued	Specifies how many calls are queued to the queue device. This number includes the current call and excludes all direct-agent calls in the queue.

Usage Notes

- The *CallQueued* event fires to indicate when a call is delivered or redirected to a hunt group or ACD split and the call queues. It also fires if the call queues to the same split with different priority.
- A *CallQueued* event never fires if a call queues to an announcement, vector announcement or trunk group. It also never fires if a call queues to the same ACD split at the same priority.

Sample Code

```
Private Sub px_CallQueued(clsCall As CallClass, queue As
String, numberQueued As Long)

      DoStatus "CallQueued Event"

End Sub
```

CallReleased

Syntax:	*CallReleased(ByVal clsCall As CallClass,ByVal EventCause As Long)*
Description:	Fires when a call on a monitored DN is released.
Response to method:	[*CallRelease*]

Parameters

clsCall	The call class representing the call that was released.
EventCause	Specifies the cause for this event. For event cause values, refer to the enumeration *enEventCause* in the *PhoneX Enumerations Guide*.

Class Settings

None.

Code Example

```
Private Sub px_CallReleased(ByVal clsCall As CallClass,
ByVal EventCause As Long)


      DoStatus "The calls have been released"
End Sub
```

CallUnHeld

Syntax:	*CallUnHeld(ByVal clsCall As CallClass)*
Description:	Fires when a call is returned from a held state to an active state. Retrieving a call can be done manually at the station by selecting the call appearance or by issuing the *CallUnHold* method.
Response to method:	[*CallUnHold*]

Parameters

clsCall	The call that has been taken off hold.

Class Settings

None.

Sample Code

```
Private Sub px_CallUnHeld(ByVal clsCall As CallClass)
      CmdHold.Caption = holdText
      If clsCall.CallState = 3 Then
            DoStatus "Call continued..."
      End If
End Sub
```

DeviceMuted

Syntax:	*DeviceMuted(ByVal DeviceDN As String)*
Description:	This event occurs when the *DeviceMute* method was successful.
Response to method:	[*DeviceMute*]

Parameters

DeviceDN	The DN that was muted.

Class Settings

None.

Sample Code

```
Private Sub px_DeviceMuted(ByVal DeviceDN As String)
      DoStatus "Device " + DeviceDN + " has been Muted."
End Sub
```

DeviceUnMuted

Syntax:	*DeviceUnMuted(ByVal DeviceDN As String)*
Description:	This event occurs when the DeviceUnMute method was successful.
Response to method:	[*DeviceUnMute*]

Parameters

DeviceDN	The DN that was unmuted.

Class Settings

None.

Sample Code

```
Private Sub px_DeviceUnMuted(ByVal DeviceDN As String)
      DoStatus "Device " + DeviceDN + " has been UnMuted."
End Sub
```

CHAPTER 10

Set Feature Methods

This chapter contains the methods that control the telephony features programmed for the phone.

In This Chapter

SetForward

Syntax:	*SetForward(ByVal clsDevice As DeviceClass) As Long*
Description:	Activates the Call Forward feature for the DeviceDN.
Returns:	Long

Parameters

clsDevice	The device class object that will receive the Call Forward status.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

SetForwardReturn	The device specified in the device object has been set/unset for the Call Forward feature.

Error Event Values

Invalid_CSTA_Device_Identifier

An invalid device identifier has been specified in the device class for either the DN or the forwarding DN.

Generic_Subscribed_Resource_Availability

Service or option not subscribed. Forwarding detection has been requested but is not enabled on the switch.

Sample Code

```
Private Sub cmdSetForward_Click()
      Dim lRtn As Long
      lRtn = px.SetForward(clsDevice)
End Sub
```

SetSendAllCalls

Syntax:	*SetSendAllCalls(ByVal clsDevice As DeviceClass) As Long*
Description:	Activates the Send All Calls feature on the specified *DeviceDN*.
Returns:	Long

Parameters

clsDevice	The device class that will receive the Send All Calls status. This will divert all calls to a specified DN as programmed in the switch.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

SetSendAllCallsReturn	The device specified in the device object has been set/unset for the Send All Calls feature.

Error Event Values

Invalid_CSTA_Device_Identifier

Service or option not subscribed. Send All Calls detection has been requested but is not enabled on the switch.

Sample Code

```
Private Sub cmdSetSendAllCalls(ByVal clsDevice As
DeviceClass)

      Dim lRtn As Long

      lRtn = px.SetSendAllCalls(clsDevice)

End Sub
```

SetBillingRate

Syntax:	*SetBillingRate(ByVal clsCall As CallClass) As Long*
Description:	Activates the Billing Rate feature (Advice of Charge) for the particular call. This feature can be requested when the call has been answered or before the call is cleared.
Returns:	Long

Parameters

clsCall	The call class that is to receive the Billing Rate status.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

SetBillingRateReturn	The call specified in the call object has been set/unset for billing rate information.

Error Event Values

Invalid_CSTA_Connection_Identifier

The connection identifier for the call is invalid.

Value_Out_Of_Range

An invalid value is specified in the request.

Invalid_Object_State

The request was specified before the call was answered.

Resource_Busy

The switch limit for unconfirmed request has been reached.

Generic_Subscribed_Resource_Availability

Service or option not subscribed. Set billing rate detection has been requested but is not enabled on the switch.

Code Example

```
Private Sub cmdSetBillingRate_Click()
      Dim lRtn As Long
      lRtn = px.SetBillingRate(clsCall)
End Sub
```

SetMessageWaiting

Syntax:	*SetMessageWaiting(ByVal clsDevice As DeviceClass) As Long*
Description:	Activates the Message Waiting feature on the monitored DN.
Returns:	Long

Parameters

clsDevice	The device class that will receive the Message Waiting status.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

SetMessageWaitingReturn	The Message Waiting feature for the device specified in the device class has been set/unset.

Error Event Values

Invalid_CSTA_Device_Identifier

The device identifier specified is invalid.

Sample Code

```
Private Sub cmdSetMessageWaiting_Click()
      Dim lRtn As Long
      lRtn = px.SetMessageWaiting(clsDevice)
End Sub
```

CHAPTER 11

Set Feature Events

This chapter contains the events that PhoneX will return from setting features.

In This Chapter

SetForwardReturn

Syntax:	*SetForwardReturn(ByVal clsDevice As DeviceClass)*
Description:	Fires when the Call Forward feature has been set/unset.
Response to method:	[*SetForward*]

Parameters

clsDevice	The device that receives the *SetForward* method.

Class Settings

StatusFWD	This property is of type Boolean. It will be set to True if the Call Forward feature is activated and False if it is turned off.

Sample Code

```
Private Sub px_SetForwardReturn(ByVal clsDevice _
      As DeviceClass)

      If clsDevice.StatusFWD = True Then
            LedFWD.BackColor = vbGreen
      Else
            LedFWD.BackColor = vbGray
      End If
End Sub
```

SetSendAllCallsReturn

Syntax:	*SetSendAllCallsReturn(ByVal clsDevice As DeviceClass)*
Description:	Fires when the Send All Calls feature has been set/unset.
Response to method:	[*SetSendAllCalls*]

Parameters

clsDevice	The device that receives the *SetSendAllCalls* method.

Class Settings

StatusSAC	This property is of type Boolean. It will be set to True if the Send All Calls feature is activated and False if it is turned off.

Sample Code

```
Private Sub px_SetSendAllCallsReturn(ByVal _
      clsDevice As DeviceClass)

      If clsDevice.StatusSAC = True Then
            LedSAC.BackColor = vbGreen
      Else
            LedSAC.BackColor = vbGray
      End If
End Sub
```

SetBillingRateReturn

Syntax:	*SetBillingRateReturn(ByVal clsDevice As DeviceClass)*
Description:	Fires when the Billing Rate feature (Advice of Charge) has been set/unset.
Response to method:	[*SetBillingRate*]

Parameters

clsDevice	The device class that receives the *SetBillingRate* method.

Class Settings

BillingRate	This property is of type string. This will display the billing rate for the call in progress.

Sample Code

```
Private Sub px_SetBillingRateReturn(ByVal clsDevice As
DeviceClass)
      DoStatus "Current Charge= " + _
      clsDevice.BillingRate + "per minute"
End Sub
```

SetMessageWaitingReturn

Syntax:	*SetMessageWaitingReturn(ByVal clsDevice As DeviceClass)*
Description:	Fires when the Message Waiting feature has been set/unset.

Response to method: [*SetMessageWaiting*]

Parameters

clsDevice	The device that receives the *SetMessageWaiting* method.

Class Settings

StatusMWT	This property is of type Boolean. It will be set to True if the Message Waiting feature is activated and False if it is turned off.

Sample Code

```
Private Sub px_SetMessageWaitingReturn(ByVal _
      clsDevice As DeviceClass)

      If clsDevice.StatusMWT = True Then
            LedMWT.BackColor = vbRed
      Else
            LedMWT.BackColor = vbGray
      End If
End Sub
```

CHAPTER 12

Agent Methods

This chapter contains the methods that may be used in order to agent login or logout. The state changes may also be set using these methods.

In This Chapter

AgentLogin

Syntax:	*AgentLogin(ByVal clsAgent As AgentClass) As Long*
Description:	Logs an agent into a specified split/skill at the specified DN.
Returns:	Long

Parameters

clsAgent	The agent class that contains the relevant information to log in an agent.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

AgentLoggedIn	This event will fire when the agent login performed was successful.

Error Event Values

Generic_Unspecified

An attempt to log in the agent to a split/skill was unsuccessful due to the agent not being a member, or an attempt to login with an invalid password was performed.

Generic_Operation

An attempt to login an agent that was already logged in.

Object_Not_Known

One or all of the DN, *SplitSkill* and *AgentID* fields are empty.

Invalid_CSTA_Device_Identifier

The DN specified is invalid

Generic_State_Incompatibility

The agent station is in maintenance busy or out of service.

Invalid_Object_State

The agent is already logged into another split.

Generic_System_Resource_Availability

The request cannot be completed due to lack of available switch resources.

Resource_Busy

An attempt to log in an ACD agent that is currently on a call.

Sample Code

```
Private Sub cmdAgentLogin_Click()
      Dim clsAgent As AgentClass
      Dim lRtn As Long

      Set clsAgent = px.AgentClasses.Add()
      clsAgent.AgentID = “4567”
      clsAgent.DN = “5054”

      lRtn = px.AgentLogin(clsAgent)
End Sub
```

AgentLogout

Syntax:	*AgentLogout(ByVal clsAgent As AgentClass) As Long*
Description:	Logs an agent out of a specified split/skill at the specified DN.
Returns:	Long

Parameters

clsAgent	The agent class that contains information to log out an agent from a split/skill.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Usage Notes

The *AgentIdentifier* is required to retrieve the appropriate *AgentClass* that relates to the agent to be logged out. Set the *AgentMode* to *amLogOut*.

Return Events (in order fired)

AgentLoggedOut	This event fires when the agent logout method was performed successfully.

Error Event Values

Generic_Unspecified

An attempt to log out the agent from a split/skill was unsuccessful as the agent was already logged out.

Sample Code

```
Private Sub cmdAgentLogout_Click()
      Dim clsAgent As AgentClass
      Dim lRtn As Long

      Set clsAgent = px.AgentClasses.Item(agentIdentTxt)
      clsAgent.AgentMode = amLogOut
      lRtn = px.AgentLogout(clsAgent)
 End Sub
```

AgentSetState

Syntax:	*AgentSetState(ByVal clsAgent As AgentClass) As Long*
Description:	Sets the agent state for the specified monitored DN.
Returns:	Long

Parameters

clsAgent	The agent class containing the information of the state change to be made.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

AgentStateReturn	This event fires when the *AgentSetState* method was successful.

Error Event Values

Value_Out_Of_Range

The WorkMode is invalid for the provided *AgentMode*.

Invalid_Feature

The feature is not available for the particular *SplitSkill*.

Invalid_Object_Type

The reason code was specified, but the agent work mode is not in WM_AUX_WORK or the agent mode is not *amLogOut* as defined by the enumerations *enAgentWorkMode* and *enAgentMode* in the *PhoneX Enumerations Guide*.

Sample Code

```
Private Sub cmdAgentSetState_Click()
      Dim lRtn As Long
      Dim agtClass As AgentClass

      Set agtClass = px.AgentClasses.Item(agtIdentTxt)
      If Not agtClass Is Nothing Then
            AgtClass.AgentMode = enAgentMode.amNotReady
            AgtClass.WorkMode = enAgentMode.wmAux
            lRtn = px.AgentSetState(agtClass)
      End If
End Sub
```

CHAPTER 13

Agent Events

This chapter contains the agent events that return when the login, logout or state changes of an agent are successful.

In This Chapter

AgentLoggedIn

Syntax:	*AgentLoggedIn(ByVal clsAgent As AgentClass)*
Description:	Fires when an agent logs into a split/skill at a monitored DN.
Response to method:	[*AgentLogin*]

Parameters

clsAgent	The *AgentClass* object that has had the log in.

Class Settings

AgentMode	The agent mode. For agent mode values, refer to the enumeration *enAgentMode* in the *PhoneX Enumerations Guide*. The value displayed after a successful login is *amLogIn*.
WorkMode	The agent work mode. For work mode values, refer to the enumeration *enAgentWorkMode* in the *PhoneX Enumerations Guide*. The value displayed after a successful login is *wmAUX* for auxiliary work mode.
SplitAgentsLoggedIn	This will display the number of splits the agent has logged in to for non-EAS case.

Sample Code

```
Private Sub px_AgentLoggedIn(ByVal clsAgent As
AgentClass)
      ‘Change the states
      If clsAgent.AgentMode = enAgentMode.amLogIn Then
            LedINOUT.BackColor = vbGreen
            Select Case clsAgent.WorkMode
            Case enWorkMode.wmAUX:
                  LedAUX.BackColor = vbGreen
                  LedACW.BackColor = vbGray
                  LedAIn.BackColor = vbGray
                  LedMIn.BackColor = vbGray
            Case enWorkMode.wmACW
                  LedAUX.BackColor = vbGray
                  LedACW.BackColor = vbGreen
                  LedAIn.BackColor = vbGray
                  LedMIn.BackColor = vbGray
            End Select
      End If
End Sub
```

AgentLoggedOut

Syntax:	*AgentLoggedOut(ByVal clsAgent As AgentClass)*
Description:	Fires when an agent logs out of a split/skill at a monitored DN.
Response to method:	[*AgentLogout*]

Parameters

clsAgent	The AgentClass object that has had the log off.

Class Settings

AgentMode	The agent mode. For agent mode values, refer to the enumeration *enAgentMode* in the *PhoneX Enumerations Guide*. The value displayed after a successful logout is *amLogOut*.
SplitAgentsLoggedIn	This displays the number of splits the agent has logged out for Non-EAS.

Sample Code

```
Private Sub px_AgentLoggedOut(ByVal clsAgent As
AgentClass)
      ‘Change the state to log out
      If clsAgent.AgentMode =EnAgentMode.amLogOut Then
            LedINOUT.BackColor = vbGray
            LedAUX.BackColor = vbGray
            LedACW.BackColor = vbGray
            LedAIn.BackColor = vbGray
            LedMIn.BackColor = vbGray
      End If
End Sub
```

AgentStateReturn

Syntax:	*AgentStateReturn(ByVal clsAgent As AgentClass)*
Description:	Fires after a state change has occurred for an agent.
Response to method:	[*AgentSetState*]

Parameters

clsAgent	The *AgentClass* object that has had the change in state.

Class Settings

AgentMode	The agent mode. For agent mode values, refer to the enumeration *enAgentMode* in the *PhoneX Enumerations Guide*.
WorkMode	The agent work mode. For work mode values, refer to the enumeration *enAgentWorkMode* in the *PhoneX Enumerations Guide*.

Sample Code

```
Private Sub px_AgentStateReturn(ByVal clsAgent As
AgentClass)
      ‘Change the states
      If clsAgent.AgentMode = enAgentMode.amReady Then
            LedINOUT.BackColor = vbGreen
            Select Case clsAgent.WorkMode
            Case enWorkMode.wmAUX:
                  LedAUX.BackColor = vbGreen
                  LedACW.BackColor = vbGray
                  LedAIn.BackColor = vbGray
                  LedMIn.BackColor = vbGray
            Case enWorkMode.wmACW
                  LedAUX.BackColor = vbGray
                  LedACW.BackColor = vbGreen
                  LedAIn.BackColor = vbGray
                  LedMIn.BackColor = vbGray
            End Select
      End If
End Sub
```

CHAPTER 14

Query Methods

This chapter includes the PhoneX methods that perform querying.

In This Chapter

QueryACDSplit

Syntax:	*QueryACDSplit(ByVal DN As String) As Long*
Description:	Queries an ACD split to find out the number of logged-in agents, number of agents available to receive calls, and the number of calls in queue. Note: The number of queued calls does not include direct agent calls.
Returns:	Long

Parameters

DN	This parameter must be a valid ACD split extension in order to perform this query.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events

QueryACDSplitReturn	Returns with updated information on the agent through the agent class.

Error Event Values

Invalid_CSTA_Device_Identifier

The device specified in DN is invalid.

Sample Code

```
Private Sub cmdQueryACDSplit_Click()
      Dim lRtn As Long
      lRtn = px.QueryACDSplit(mySplitDN)
End Sub
```

QueryAgentLogin

Syntax:	*QueryAgentLogin(ByVal DN As String) As Long*
Description:	Requests the extensions numbers of each agent logged in to an ACD split.
Returns:	Long

Parameters

DN	This parameter must be a valid ACD split extension in order to perform this query.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events

QueryAgentLoginReturn	Returns with updated information on the agent through the agent class.

Error Event Values

Invalid_CSTA_Device_Identifier

The device specified in DN is invalid.

Sample Code

```
Private Sub cmdQueryAgentLogin_Click()
      Dim lRtn As Long
      lRtn = px.QueryAgentLogin(mySplitDN)
End Sub
```

QueryAgentState

Syntax:	*QueryAgentState(ByVal clsAgent As AgentClass) As Long*
Description:	Requests the agent status of the specified DN for the specified split/skill.
Returns:	Long

Parameters

clsAgent	The agent class that is to be queried for its agent state.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events

QueryAgentStateReturn	Returns with updated information on the agent through the agent class.

Error Event Values

Invalid_CSTA_Device_Identifier

The split/skill specified in the agent class is invalid.

Sample Code

```
Private Sub cmdQueryAgentState_Click()
      Dim lRtn As Long
      Dim agtClass As AgentClass

      Set agtClass = px.AgentClasses.Item(agentIdentTxt)
      If Not agtClass Is Nothing Then
            lRtn = px.QueryAgentState(agtClass)
      End If
End Sub
```

QueryCallClassifier

Syntax:	*QueryCallClassifier() As Long*
Description:	Requests the number of “idle” and “in-use” ports (eg. TN744). The “in-use” number is a snapshot of the call classifier port usage.
Returns:	Long

Parameters

None.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events

QueryCallClassifierReturn	Returns with updated information on number of busy and idle call classifiers.

Error Event Values

None.

Sample Code

```
Private Sub cmdQueryCallClassifier_Click()
      Dim lRtn As Long
      lRtn = px.QueryCallClassifier()
End Sub
```

QueryDeviceInfo

Syntax:	*QueryDeviceInfo(ByVal DN As String) As Long*
Description:	Requests information about the class and type of device. The class is one of voice, data, image or other. The type attribute is one of station, ACD, ACD Group or other.
Returns:	Long

Parameters

DN	This parameter contains the on-switch station extension.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

QueryDeviceInfoReturn	Returns with information that relates to the DN that was queried.

Error Event Values

Invalid_CSTA_Device_Identifier

The device that was specified in the DN parameter is invalid.

Sample Code

```
Private Sub cmdQueryDeviceInfo_Click()
      Dim lRtn As Long
      lRtn = px.QueryDeviceInfo(theDNTxt)
End Sub
```

QuerySendAllCalls

Syntax:	*QuerySendAllCalls(ByVal DN As String) As Long*
Description:	Requests the status of the Send All Calls feature for DN. The feature will be expressed as False if the DN does not have a coverage path.
Returns:	Long

Parameters

DN	This parameter contains the on-switch station extension that supports the Send All Calls feature.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

QuerySendAllCallsReturn	Returns the status of the Send All Calls feature for the queried DN.

Error Event Values

Invalid_CSTA_Device_Identifier

The device that was specified in the DN parameter is invalid.

Sample Code

```
Private Sub cmdQuerySendAllCalls_Click()
      Dim lRtn As Long
      lRtn = px.QuerySendAllCalls(myDN)
 End Sub
```

QueryForward

Syntax:	*QueryForward(ByVal DN As String) As Long*
Description:	Requests the status of the Call Forward feature for DN.
Returns:	Long

Parameters

DN	This parameter contains the on-switch station extension that supports the Call Forward feature.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

QueryForwardingReturn	Returns the status of the Call Forward feature for the queried DN.

Error Event Values

Invalid_CSTA_Device_Identifier

The device that was specified in the DN parameter is invalid.

Sample Code

```
Private Sub cmdQueryForward_Click()
      Dim lRtn As Long
      lRtn = px.QueryForward(myDN)
End Sub
```

QueryMessageWaiting

Syntax:	*QueryMessageWaiting(ByVal DN As String) As Long*
Description:	Requests the status of the Message Waiting feature for the DN.
Returns:	Long

Parameters

DN	This parameter contains the on-switch station extension that supports the Message Waiting feature.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

QueryMessageWaitingReturn	Returns the status of the Message Waiting feature for the DN that was queried and the application that turn the indicator on.

Error Event Values

Invalid_CSTA_Device_Identifier

The device that was specified in the DN parameter is invalid.

Sample Code

```
Private Sub cmdQueryMessageWaiting_Click()
      Dim lRtn As Long
      lRtn = px.QueryMessageWaiting(myDN)
End Sub
```

QueryTimeOfDay

Syntax:	*QueryTimeOfDay() As Long*

Description:	Requests the current time and date from the switch. The time is in 24-hour format and includes minutes and seconds. The date will return with year, month and day values.
Returns:	Long

Parameters

None.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

QueryTimeOfDayReturn	Returns the year, month, day, hour, minutes and seconds.

Error Event Values

None.

Sample Code

```
Private Sub cmdQueryTimeOfDay_Click()
      Dim lRtn As Long
      lRtn = px.QueryTimeOfDay()
End Sub
```

QueryTrunkGroup

Syntax:	*QueryTrunkGroup(ByVal DN As String) As Long*
Description:	Requests the status of the specified trunk group <*TAC*>. Requests the number of idle trunks and the number of in-use trunks. The sum of the idle and in-use trunks provides the number of trunks in service.
Returns:	Long

Parameters

DN	This parameter specifies a valid trunk group access code.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

QueryTrunkGroupReturn	Returns the status of used and idle trunks for the trunk group access code supplied.

Error Event Values

Invalid_CSTA_Device_Identifier

The DN was specified with an invalid device identifier.

Sample Code

```
Private Sub cmdQueryTrunkGroup_Click()
      Dim lRtn As Long
      lRtn = px.QueryTrunkGroup(trunkDN)
End Sub
```

QueryStationStatus

Syntax:	*QueryStationStatus(ByVal DN As String) As Long*
Description:	Requests information on the busy or idle state of the specified station DN. A busy state relates to an active call on the station. The idle state is returned if the station is not on any call.
Returns:	Long

Parameters

DN	This parameter specifies a valid station device.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

QueryStationStatusReturn	Returns the idle and/or busy state for the station.

Error Event Values

Invalid_CSTA_Device_Identifier

The DN was specified with an invalid device identifier.

Sample Code

```
Private Sub cmdQueryStationStatus_Click()
      Dim lRtn As Long
      lRtn = px.QueryStationStatus(stationDN)
End Sub
```

QueryUCID

Syntax:	*QueryUCID(ByVal CallID As Long) As Long*
Description:	Requests the UCID (Universal Call ID) for the Call ID associated with the call. This method may be called at anytime during the lifetime of the call.
Returns:	Long

Parameters

CallID	This parameter specifies a valid call ID that is currently on a call.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

QueryUCIDReturn	Returns the UCID for the call.

Error Event Values

Invalid_CSTA_Call_Identifier

The specified *CallID* is invalid.

Invalid_Feature

The switch software does not support this functionality. The switch software version may be earlier than Release 6.

Sample Code

```
Private Sub cmdQueryUCID_Click()
      Dim lRtn As Long
      lRtn = px.QueryUCID(currCallID)
End Sub
```

CHAPTER 15

Query Events

This chapter contains information related to the events that return when the querying is successful.

In This Chapter

QueryACDSplitReturn

Syntax:	*QueryACDSplitReturn(ByVal clsAgent As AgentClass)*
Description:	Returns ACD split/skill information.
Response to method:	[*QueryACDSplit*]

Parameters

clsAgent	The *AgentClass* object that contains the updated information as a result of the query.

Class Settings

SplitCallsInQueue	This contains the number of calls waiting in queue.
SplitAgentsAvailable	The number of agents currently available to receive calls.
SplitAgentsLoggedIn	The number of agents currently logged into this split/skill.

Sample Code

```
Private Sub px_QueryACDSplitReturn(ByVal clsAgent As
AgentClass)

      DoStatus "Calls Waiting = " +
clsAgent.SplitCallsInQueue

      DoStatus "Available Agents = " +
clsAgent.SplitAgentsAvailable

      DoStatus "Agents Logged In = " +
clsAgent.SplitAgentsLoggedIn

End Sub
```

QueryAgentLoginReturn

Syntax:	*QueryAgentLoginReturn(ByVal clsAgent As AgentClass)*
Description:	Returns the extension number for each agent logged in to a split/skill.
Response to method:	[*QueryAgentLogin*]

Parameters

clsAgent	The *AgentClass* object that contains the updated information as a result of the query.

Class Settings

DN	This contains the extension number of the agent logged in to this split/skill.

SplitSkill	The split/skill extension number that the agent was logged in to.

Sample Code

```
Private Sub px_QueryAgentLoginReturn(ByVal clsAgent As
AgentClass)
      DoStatus "Agent DN = " + clsAgent.DN
      DoStatus "Agent Split/Skill = " +
clsAgent.SplitSkill
End Sub
```

QueryAgentStateReturn

Syntax:	*QueryAgentStateReturn(ByVal clsAgent As AgentClass)*
Description:	Returns the agent status of the specified DN for the specified split/skill.
Response to method:	[*QueryAgentState*]

Parameters

clsAgent	The *AgentClass* object that contains the updated information as a result of the query.

Class Settings

AgentState	The agent state. For agent state values, refer to the enumeration *enAgentState* in the *PhoneX Enumerations Guide*.
WorkMode	The agent work mode. For agent mode values, refer to the enumeration *enAgentWorkMode* in the *PhoneX Enumerations Guide*. This field is only defined if the *AgentState* value is asReady.
ReasonCode	The reason code for the appropriate agent state. The meanings for the reason should be defined in the controlling application.

Sample Code

```
Private Sub px_QueryAgentStateReturn(ByVal clsAgent As
AgentClass)
      DoStatus "Agent State = " +
Cstr(clsAgent.AgentState)
      DoStatus "Agent WorkMode = " +
Cstr(clsAgent.WorkMode)
      DoStatus "Agent ReasonCode = " +
Cstr(clsAgent.ReasonCode)
End Sub
```

QueryCallClassifierReturn

Syntax:	*QueryCallClassifierReturn(ByVal Busy As Long, ByVal Idle As Long)*
Description:	Returns the number of “idle” and “in-use” ports. The “in-use” number is a snapshot of the call classifier port usage.
Response to method:	[*QueryCallClassifier*]

Parameters

Busy	The number of “in-use” ports.
Idle	The number of “idle” ports.

Class Settings

None.

Sample Code

```
Private Sub px_QueryCallClassifierReturn(ByVal _
      Busy As Long, ByVal Idle As Long)
      DoStatus "Busy Ports = " + Cstr(Busy)
      DoStatus "Idle Ports = " + Cstr(Idle)
End Sub
```

QueryDeviceInfoReturn

Syntax:	*QueryDeviceInfoReturn(ByVal DN As String, ByVal DevType As Long, ByVal Class As Long, ByVal ExtnClass As Long, ByVal AssocDN As String, ByVal AssocClass As Long, ByVal Name As String)*
Description:	Returns information about the device DN.
Response to method:	[*QueryDeviceInfo*]

Parameters

DN	The specified DN queried for information.
DevType	The device type for the DN. This could be any one of station, ACD, ACD Group or other.
Class	The class contains one of the following categories that are void, data, image or other.
ExtnClass	The G3 extension class for the device queried.

AssocDN	If the device specified in the request is a physical device of a logical agent who is logged in, the logical ID of that agent is returned in this parameter. If the device specified in the request is the logical ID of a logged-in agent, the physical device ID of that agent is returned in this parameter. Otherwise, a null string is returned.
AssocClass	The G3 extension class for the *AssocDN*.
Name	This field contains the information related to the name of the device specified as programmed in the switch.

Class Settings

None.

Sample Code

```
Private Sub px_QueryDeviceInfoReturn(ByVal _

DN As String, ByVal DevType As Long, ByVal Class As Long,
ByVal ExtnClass As Long, ByVal AssocDN As String, ByVal
AssocClass As Long, ByVal Name As String)


      DoStatus "Device = " + DN

      DoStatus "Device Type = " + Cstr(DevType)

      DoStatus "Class = " + Cstr(Class)

      DoStatus "ExtnClass = " + Cstr(ExtnClass)

      If Not AssocDN Is Empty Then

            DoStatus "AssocDN = " + AssocDN

            DoStatus "AssocClass = " + Cstr(AssocClass)

      End If

      DoStatus "Name = " + Name

End Sub
```

QueryTimeOfDayReturn

Syntax:	*QueryTimeOfDayReturn(ByVal Year As Long, ByVal Month As Long, ByVal Day As Long, ByVal Hour As long, ByVal Minute As Long, ByVal Second As Long)*
Description:	Returns time-of-day information from the PBX.
Response to method:	[*QueryTimeOfDay*]

Parameters

Year	The present year as defined in the switch.
Month	The present month as defined in the switch.
Day	The present day as defined in the switch.

Hour	The present hour as defined in the switch.
Minute	The present minute as defined in the switch.
Second	The present second as defined in the switch.

Class Settings

None.

Sample Code

```
Private Sub px_QueryTimeOfDayReturn(ByVal Year As Long,
ByVal Month As Long, ByVal Day As Long, ByVal Hour As long,
ByVal Minute As Long, ByVal Second As Long)


      Dim TimeStamp As String
      Dim DateStamp As String


      TimeStamp = cstr(Hour) + ":" + cstr(Minute) + ":"
+ cstr(Second)
      DateStamp = cstr(Month) + "/" + cstr(Day) + "/" +
Cstr(Year)
End Sub
```

QueryTrunkGroupReturn

Syntax:	*QueryTrunkGroupReturn(ByVal DN As String, ByVal Used As Long, ByVal Idle As Long)*
Description:	Returns trunk group information for <*TAC*>.
Response to method:	[*QueryTrunkGroup*]

Parameters

DN	This is the trunk group access code that was queried.
Used	The number of ‘in-use’ trunks.
Idle	The number of ‘idle’ trunks.

Class Settings

None.

Sample Code

```
Private Sub px_QueryTrunkGroupReturn(ByVal DN As String,
ByVal Used As Long, ByVal Idle As Long)


      DoStatus "Trunk Group = " + DN
      DoStatus "Used trunks = " + cstr(Used)
      DoStatus "Idle trunks = " + cstr(Idle)
End Sub
```

QueryStationStatusReturn

Syntax:	*QueryStationStatusReturn(ByVal DN As String, ByVal Status As Long)*
Description:	Returns busy/idle status of the specified device.
Response to method:	[*QueryStationStatus*]

Parameters

DN	The station device that was queried.
Status	The status of the station. This is a Boolean value of if the station is busy or False if the station is idle.

Class Settings

None.

Sample Code

```
Private Sub px_QueryStationStatusReturn(ByVal DN As
String, ByVal Status As Long)


      DoStatus "Station DN = " + DN
      If Status = True Then
            LedBusyStatus.BackColor = vbGreen
      Else
            LedBusyStatus.BackColor = vbGray
      End If
End Sub
```

QuerySendAllCallsReturn

Syntax:	*QuerySendAllCallsReturn(ByVal DN As String, ByVal Status As Long)*
Description:	Returns the status of the Send All Calls feature for the specified device.
Response to method:	[*QuerySendAllCalls*]

Parameters

DN	The station DN that this query was made upon.
Status	The status of the Send All Calls feature. It is set to True if the Send All Calls feature was turned on and False if it is off.

Class Settings

None.

Sample Code

```
Private Sub px_QuerySendAllCallsReturn(ByVal DN As
String, ByVal Status As Long)


      If Status = True Then
            DoStatus "SAC is active"
            LedSAC.BackColor = vbGreen
      Else
            DoStatus "SAC is not active"
            LedSAC.BackColor = vbGray
      End If
End Sub
```

QueryForwardingReturn

Syntax:	*QueryForwardingReturn(ByVal DN As String, ByVal ForwardState As Long, ByVal ForwardDN As String)*
Description:	Returns the status of the Call Forward feature for specified device.
Response to method:	[*QueryForwarding*]

Parameters

DN	This is the station DN that was queried.

ForwardState	The returned value for this parameter is True (forwarding is active) or False (forwarding is not active).
ForwardDN	This parameter stores the forwarding destination number.

Class Settings

None.

Sample Code

```
Private Sub px_QueryForwardingReturn(ByVal DN As String,
ByVal ForwardState As Long, ByVal ForwardDN As String)


      If ForwardState = True Then
            LedFWD.BackColor = vbRed
            DoStatus "FWD is active on " + ForwardDN
      Else
            LedFWD.BackColor = vbGray
            DoStatus "FWD is not active"
      End If
End Sub
```

QueryMessageWaitingReturn

Syntax:	*QueryMessageWaitingReturn(ByVal DN As String, ByVal MWState As Long, ByVal MWActivator As Long)*
Description:	Returns the status of the Message Waiting feature for the specified device.
Response to method:	[*QueryMessageWaiting*]

Parameters

DN	This parameter returns the on-switch station extension number that was queried.
MWState	This parameter returns True if the Message Waiting feature is on or False if it is off.
MWActivator	This parameter returns the application that activated the Message Waiting feature.

Class Settings

None.

Code Example

```
Private Sub px_QueryMessageWaitingReturn(ByVal DN As
String, ByVal MWState As Long, ByVal MWActivator As Long)


      DoStatus "Station DN = " + DN
      If MWState = True Then
            LedMWI.BackColor = vbRed
            DoStatus "Activator = " + cstr(MWActivator)
      Else
            LedMWI.BackColor = vbGray
      End If
End Sub
```

QueryUCIDReturn

Syntax:	*QueryUCIDReturn(ByVal CallID As Long, ByVal UCID As String)*
Description:	Returns the universal call ID for a specified call ID.
Response to method:	[*QueryUCID*]

Parameters

CallID	This parameter returns the call identifier for the call. This is the parameter that queried for the UCID.
UCID	The universal call identifier associated with the call ID.

Class Settings

None.

Sample Code

```
Private Sub px_QueryUCIDReturn(ByVal CallID As Long,
ByVal UCID As Long)


      DoStatus "The CallID = " + cstr(CallID) + _
      " has a UCID of " + cstr(UCID)


End Sub
```

CHAPTER 16

Snapshot Methods

This chapter contains the snapshot methods PhoneX uses to retrieve snapshots of calls or devices.

In This Chapter

SnapshotCall

Syntax:	*Snapshotcall(ByVal DN As String, ByVal CallID As Long) As Long*
Description:	Requests a snapshot of the specified <*CallID*> on the specified device.
Returns:	Long

Parameters

DN	This parameter specifies a valid station device.
CallID	This parameter specifies a valid *CallID*.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Usage Notes

This method provides information for each end point on the specified call. The information provided include Device ID, connection ID and the CSTA local connection state.

The DN parameter may be an on-switch alerting extension or a split hunt group with calls in queue. When a call is queued on more than one split hunt group, only one split hunt group extension is provided in the response to such a query. For calls alerting at various groups, the group extension is reported to PhoneX. For calls connected to a member of a group, the group member's extension is reported to PhoneX.

Return Events (in order fired)

SnapshotCallReturn	This event returns with the information in a call class.

Error Event Values

Invalid_CSTA_Call_Identifier

An invalid CallID was specified.

Invalid_CSTA_Device_Identifier

An invalid DN was specified.

Sample Code

```
Private Sub cmdSnapshotcall_Click()
      Dim lRtn As Long
      lRtn = px.Snapshotcall(myDN,callID)
End Sub
```

SnapshotDevice

Syntax:	*SnapshotDevice(ByVal DN As String) As Long*
Description:	Requests information about calls associated with a given CSTA device. The information identifies each call and indicates the CSTA local connection state for all devices on each call.
Returns:	Long

Parameters

DN	This parameter contains the valid device number.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

SnapShotDeviceReturn	This event returns the information within the device class.

Error Event Values

Invalid_CSTA_Device_Identifier

An invalid device identifier was specified in DN.

Sample Code

```
Private Sub cmdSnapShotDevice_Click()
      Dim lRtn As Long
      lRtn = px.SnapshotDevice(forwardingDN)
End Sub
```

CHAPTER 17

Snapshot Events

This chapter includes events that return upon successful snapshot requests.

In This Chapter

SnapshotCallReturn

Syntax:	*SnapshotCallReturn(ByVal clsCall As CallClass)*
Description:	Returns the snapshot for <*CallID*> on the specified device.
Response to method:	[*Snapshotcall*]

Parameters

clsCall	The call class that contains the information for the snapshot.

Class Settings

CallID	The *callID* associated with this call.
DN	The station number associated with this call.
CallState	The state the call is in. For call state values, refer to the enumeration *enCallState* in the *PhoneX Enumerations Guide*.

Sample Code

```
Private Sub px_SnapshotCallReturn(ByVal clsCall As
CallClass)
      If clsCall.CallState = enCallState.csHold Then
            LedGreen.BackColor = vbGreen
            FlashGreenLED(True)
      Else
            LedGreen.BackColor = vbGray
            FlashGreenLED(False)
      End If
End Sub
```

SnapShotDeviceReturn

Syntax:	*SnapShotDeviceReturn(ByVal clsDevice As DeviceClass)*
Description:	Returns the snapshot for the specified device.
Response to method:	[*SnapshotDevice*]

Parameters

clsDevice	The device class that contains the information for the snapshot.

Class Settings

DeviceState	The current state of the device. For device state values, refer to the enumeration *enDeviceState* in the *PhoneX Enumerations Guide*.
DeviceIdentifier	The device identifier associated with a call that is active on the device.

Sample Code

```
Private Sub px_SnapShotDeviceReturn(ByVal clsDevice As
DeviceClass)
      If clsDevice.DeviceState = _
            enDeviceStatus.staBusy Then
            ledBusyInd.BackColor = vbGreen
            DoStatus "The device " + clsDevice.DeviceDN
+ " is busy."
      Else
            LedBusyInd.BackColor = vbGray
            DoStatus "The device " + clsDevice.DeviceDN
+ " is idle."
      End If
End Sub
```

CHAPTER 18

Routing Methods

This chapter shows the routing methods supported by PhoneX to route calls that come in to a VDN.

In This Chapter

RouteRegister

Syntax:	*RouteRegister(ByVal clsDevice As DeviceClass) As Long*
Description:	Registers the controlling application as a routing server for a specific VDN.
Returns:	Long

Parameters

clsDevice	The device class that is associated with a specific VDN.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Usage Notes

The application must register itself as a routing server before any *RouteRequestService* events may be returned from the device. The controlling application may be a routing server for more than one routing device but for a specific routing device, the switch allows only one application registered as the routing server. If a routing device already has a routing server registered, subsequent use of this *RouteRegister* method will be negatively acknowledged.

Return Events (in order fired)

RouteRegistered	This event fires when the switch accepts the route registration.

Error Event Values

Outstanding_Request_Limit_Exceeded

This error indicates that the routing device already has a registered routing server.

Sample Code

```
Private Sub cmdRouteRegister_Click()
      Dim lRtn As Long
      lRtn = px.RouteRegister(clsDevice)
End Sub
```

RouteSelect

Syntax:	*RouteSelect(ByVal clsCall As CallClass) As Long*
Description:	Provides the switch with a destination in response to the *RouteRequestService* event.
Returns:	Long

Parameters

clsCall	The call class to be routed to the new destination.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

RouteUsed	This provides the actual destination (as a number) for the routing server application.
RouteEnded	Returns when the routing dialog is completed.

Error Event Values

Invalid_CSTA_Device_Identifier

An invalid routing registration request ID was specified in the device class.

Invalid_Cross_Ref_ID

An invalid route cross reference ID was specified in the device class.

Code Example

```
Private Sub cmdRouteSelect_Click()
      Dim lRtn As Long
      lRtn = px.RouteSelect(clsCall)
End Sub
```

RouteEnd

Syntax:	*RouteEnd(ByVal clsCall As CallClass) As Long*
Description:	Terminates a routing dialog for a call.
Returns:	Long

Parameters

clsCall	The call class that contains the active call that contains the VDN routing sequence to be ended.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

RouteEnded	Returns when the routing dialog is completed.

Error Event Values

Invalid_Cross_Ref_Id

An invalid cross reference ID was specified in the call class.

Sample Code

```
Private Sub cmdRouteEnd_Click()
      Dim lRtn As Long
      lRtn = px.RouteEnd(clsCall)
End Sub
```

RouteRegisterCancel

Syntax:	*RouteRegisterCancel(ByVal clsDevice As DeviceClass) As Long*
Description:	Cancels a previously registered route session. When this service request is successful, the controlling application will no longer be a routing server for the specified VDN.
Returns:	Long

Parameters

clsDevice	The device class containing the VDN information for cancellation of the route.

Return Values

For PhoneX return error values, refer to the enumeration *enPhoneXError* in the *PhoneX Enumerations Guide*.

Return Events (in order fired)

RouteRegisterCancelReturn	This event returns as a result of a successful cancellation of a routing registration.

Error Event Values

Invalid_CSTA_Device_Identifier

An invalid routing registration ID has been specified in the request.

Sample Code

```
Private Sub cmdRouteRegisterCancel_Click()
      Dim lRtn As Long
      lRtn = px.RouteRegisterCancel(clsDevice)
End Sub
```

CHAPTER 19

Routing Events

This chapter includes the events that return upon successful routing requests made by PhoneX to the switch. It also includes events that return from the switch where routing has aborted or when there is a call for routing to begin.

In This Chapter

RouteEnded

Syntax:	*RouteEnded(ByVal clsCall As CallClass)*
Description:	Returns when the routing dialog has ended.
Response to method:	[*RouteEnd*]

Parameters

clsCall	This parameter contains the call class the application sent with a *RouteEnd* request to the switch.

Class Settings

DN	This displays the VDN that got the *RouteEnd* request.
CalledDN	This contains the called DN that will be handled by the switch after terminating the routing functionality to be performed by the application.
UUI	Any user-to-user information attached to this call that entered this VDN.
CallerDigits	Any entered digits that were used.

Sample Code

```
Private Sub px_RouteEnded(ByVal clsCall As CallClass)
      DoStatus "Route Ended for " + clsCall.DN
End Sub
```

RouteRegisterAbort

Syntax:	*RouteRegisterAbort(ByVal clsDevice As DeviceClass)*
Description:	Fires when the switch aborted a routing registration session.
Response to method:	None.

Parameters

clsDevice	The device class aborted from the routing registration session.

Class Settings

DeviceDN	The VDN aborted from the routing registration session.

Sample Code

```
Private Sub px_RouteRegisterAbort(ByVal clsDevice As
DeviceClass)
      DoStatus "VDN " + clsDevice.DeviceDN + " has _
      Route registration aborted by switch."
End Sub
```

RouteRegistered

Syntax:	*RouteRegistered(ByVal clsDevice As DeviceClass)*
Description:	Fires when the request to register the application as a routing application is successful.
Response to method:	[*RouteRegister*]

Parameters

clsDevice	The device class that contains the VDN information for the routing application.

Class Settings

DeviceDN	The VDN number that was successfully registered for the routing application.

Sample Code

```
Private Sub px_RouteRegistered(ByVal clsDevice As
DeviceClass)
      DoStatus "Route registration for the VDN " + _
      clsDevice.DeviceDN + " has been successful."
End Sub
```

RouteRegisterCanceled

Syntax:	
Description:	Fires when the
Response to method:	

Parameters

Class Settings

Sample Code

RouteRequestService

Syntax:	*RouteRequestService(ByVal clsCall As CallClass)*
Description:	Fires when the switch requests the controlling application to select a route for the current call. The application, if connected to a database, may use certain parameters in the call class to determine which route the call will be routed.
Response to method:	None.

Parameters

clsCall	The call class that contains the call to be routed.

Class Settings

CallerDN	The DN associated with the call to be routed.
UUI	Any user-to-user information strings included with the call.
CallerDigits	Any user-entered digits included with the call.
CallID	The call ID of the call to be routed.
UCID	Any universal call ID associated with the *CallID*.

Sample Code

```
Private Sub px_RouteRequestService(ByVal clsCall As
CallClass)
      Dim lRtn As Long


      If clsCall.CallerDigits = "123456" Then
            lRtn = px.RouteSelect(clsCall)
      Else
            lRtn = px.RouteEnd(clsCall)
      End If
End Sub
```

RouteUsed

Syntax: *RouteUsed(ByVal clsCall As CallClass)*

Description:	Fires when the switch has provided the routing server application with the destination for a call.
Return event:	[*RouteSelect*]

Parameters

clsCall	The call class containing the destination that was returned from the switch.

Class Settings

CalledDN	The destination DN the call is being routed to.
CallerDN	The DN where the call originated from.
DN	The VDN that performed the routing.

Sample Code

```
Private Sub px_RouteUsed(ByVal clsCall As CallClass)
      DoStatus "Route Used = " + clsCall.CalledDN
      DoStatus "Originated = " + clsCall.CallerDN
      DoStatus "Route Source = " + clsCall.DN
End Sub
```

CHAPTER 20

Language Properties

This chapter includes the property used for language control in PhoneX:

In This Chapter

Language

Syntax:	*Language*
Description:	This specifies the language type to be used with PhoneX.

Usage Notes

This property has a default value set to English. There are many languages that can be used with PhoneX. Please check with your supplier for the availability of the language of your choice.

CHAPTER 21

Language Methods

This chapter includes the methods that allow language capability within PhoneX.

In This Chapter

GetFontObject

Syntax:	*GetFontObject(ByVal FontObjectNumber As Long) As StdFont*
Description:	Requests the font type to be used with the selected language.
Returns:	*StdFont*

Parameters

FontObjectNumber	These are the selected types of font objects that PhoneX can use for a specific language.

Return Values

This returns the font type associated with the *FontObjectNumber*.

Usage Notes

This method will retrieve the relevant font object associated with the *FontObjectNumber*. The relevant font object is defined by the language controls. It returns the default font object if no language has been selected.

Return Events

None.

Sample Code

```
Private Sub cmdGetFontObject_Click()
      Dim font As StdFont
      Set font = px.GetFontObject(1)
      DoStatus “Font Name = ” & font.Name
End Sub
```

GetStringValue

Syntax:	*GetStringValue(ByVal StringID As Long) As String*
Description:	Retrieves the string associated with the *StringID* value. This is the relevant string that is translated into the specified language.
Returns:	String

Parameters

StringID	The number associated with the string to be returned.

Return Values

The string associated with the *StringID*.

Return Events

None.

Sample Code

```
Private Sub GetResourceStrings()

      CmdButton1.Caption = Me.px.GetStringValue(4032)

End Sub
```

GetStringValueEx

Syntax:	*GetStringValueEx(ByVal StringID As Long, ByVal Param1 As String, Optional ByVal Param2 As String, Optional ByVal Param3 As String, Optional ByVal Param4 As String) As String*
Description:	Retrieves the string associated with the *StringID* plus any additional, user-specific string information.
Returns:	String

Parameters

StringID	The number associated with the string to be returned.
Param1	The user-defined component of the string.
Param2	The second (optional) user-defined component of the string.
Param3	The third (optional) user-defined component of the string.
Param4	The fourth (optional) user-defined component of the string.

Return Values

The parameters are designated within the string by ‘%%#’, where # represents a number between one and four. These numbers relate to the parameter numbers, eg. %%1 relates to *Param1*.

Return Events

None.

Sample Code

If the string 4033 has the following text: “Incoming call from %%1 alerting at device %%2”. This will translate to “Incoming call from 8850 alerting at device 8650”.

```
Private Sub GetResourceStrings()

      TxtText1.Text = Me.px.GetStringValueEx(4033,
“8850”, “8650”)

End Sub
```

CHAPTER 22

Miscellaneous Methods

This chapter includes the miscellaneous methods supported by PhoneX. These methods are not directly related to telephony but are essential for the telephony functionality to work correctly.

In This Chapter

NumberUnformatNumber

Syntax:	*NumberUnformatNumber(ByVal Number As String) As String*
Description:	Unformats any extra dialable characters that appear with the CLI.
Returns:	String

Parameters

Number	The number to be unformatted

Return Values

The received number after unformatting has been done for the appropriate dialable characters.

Usage Notes

This method should be used before performing a call since the file that stores the numbers may have special character formatting. For example, in a database, a phone number is stored as (0800) 23-4-5678. Passing it through this method removes the brackets, spaces, dashes and any other invalid characters. The number is converted to 08002345678, which is accepted by the switch.

Return Events

None.

Error Event Values

None.

Sample Code

```
Private Sub cmdMakeCall_Click()
      Dim lRtn As Long
      Dim clsCall As CallClass

      Set clsCall = px.ActiveCallClasses.Add()
      If Not clsCall Is Nothing Then
            clsCall.CallerDN = "8572"
            clsCall.CalledDN =
NumberUnformatNumber("(0800) 23-4-5678")
            lRtn = px.CallDial(clsCall)
      End If
End Sub
```

NumberCheckDialableNumber

Syntax:	*NumberCheckDialableNumber(ByVal Number As String) As Long*
Description:	Checks to see if the number to be dialled is a valid number.
Returns:	Long

Parameters

Number	The number to be checked to determine that it contains only dialable characters.

Return Values

If the number contains only dialable characters, the function returns 0. If not, the function returns the location of the first non-dialable character. If the number is too long, the length+1 is returned.

Usage Notes

This method performs a check to ensure the number to be dialled is acceptable to the switch. It does not check for invalid destination numbers. Any invalid characters (eg. {[}]`) will cause this method to return the location of the first non-dialable character.

Return Events

None.

Error Event Values

None.

Sample Code

```
Private Sub cmdMakeCall_Click()
      Dim lRtn As Long
      Dim isDialable As Long
      Dim clsCall As CallClass
      Set clsCall = px.ActiveCallClasses.Add()
      If Not clsCall Is Nothing Then
            clsCall.CallerDN = "8572"
            isDialable =
px.NumberCheckDialableNumber(clsCall.CalledDN)
            If isDialable = 0 Then
                  lRtn = px.CallDial(clsCall)
            End If
      End If
End Sub
```

NumberGetDialableCharacters

Syntax:	*NumberGetDialableCharacters() As String*
Description:	Retrieves the dialable character set PhoneX is currently using.
Returns:	String

Parameters

None.

Return Values

The method returns a string variable that contains a list of all the characters considered as valid, dialable characters.

Usage Notes

This method may be used anytime when the dialable character set needs to be displayed or when the controlling application needs to use the dialable character set to perform its own checks.

Return Events

None.

Error Event Values

None.

Sample Code

```
Private Sub cmdGetDialableCharacterSet_Click()
      Dim dialableString As String

      dialableString = px.NumberGetDialableCharacters()
      DoStatus dialableString
End Sub
```

NumberSetDialableCharacters

Syntax:	*NumberSetDialableCharacters(ByVal CharacterSet As String) As Long*
Description:	Changes the dialable character set.
Returns:	Long

Parameters

CharacterSet	This string variable replaces the current valid dialable character set.

Return Values

CerrorNoError	This returns if the character set has been accepted.

Return Events

None.

Error Event Values

None.

Sample Code

```
Private Sub cmdChangeDialableCharacterSet_Click()
      Dim lRtn As Long
      Dim newSet As String

      newSet = "1234567890*#"
      lRtn = px.NumberSetDialableCharacters(newSet)
End Sub
```

VersionGetPhoneXVersion

Syntax:	*VersionGetPhoneXVersion() As String*
Description:	Gets the current version information for PhoneX.
Returns:	String

Parameters

None.

Return Values

The method returns the version of the control. This will be in the format of “Major Version”, “Minor Version”, “Fix Number” i.e. “2|14|5”

Return Events

None.

Error Event Values

None.

Sample Code

```
Private Sub cmdGetPXVersion_Click()
      Dim vString As String
      vString = px.VersionGetPhoneXVersion()
End Sub
```

AboutBox

Syntax:	*AboutBox()*
Description:	Retrieves an About Box that displays version information for PhoneX, the Telephony Server and the switch software.
Returns:	None.

Parameters

None.

Return Values

This method will display the About Box for PhoneX.

Return Events

None.

Error Event Values

None.

Sample Code

```
Private Sub cmdAboutBox_Click()
      px.AboutBox
End Sub
```

CHAPTER 23

Miscellaneous Events

This chapter contains the miscellaneous events supported by PhoneX.

In This Chapter

TSError

Syntax:	*TSError(ByVal clsError As ErrorClass)*
Description:	Fires when a problem has occurred with PhoneX.
Response to method:	None.

Parameters

clsError	The error class that is returned. For more information, refer to the chapter, Class Structures.

Class Settings

None.

Sample Code

```
Private Sub px_TSError(ByVal clsError As ErrorClass)

      FrmError.ErrorCode = clsError.ErrorCode
      FrmError.InvokeID = clsError.InvokeID
      FrmError.ErrorType = clsError.ErrorType
      FrmError.ErrorLevel = clsError.ErrorLevel
      FrmError.ErrorText = clsError.ErrorContext
      FrmError.ResolutionText = clsError.ErrorDevice
      FrmError.Show vbModal

End Sub
```

CHAPTER 24

Control Properties

This chapter defines the control properties used by PhoneX.

In This Chapter

ApplicationName

Syntax: *ApplicationName as string*

Description: The ApplicationName property is used by the PhoneX control when issuing the login request to Avaya CT and when requesting a license from the License Server.

Usage Notes

If this property is set by the controlling application, then the License Server must have an installed license of the same name. If there is no license of the same name then the PhoneX control will return an error indicating that a runtime license can not be found.

The default setting for this property is blank. When blank, the License Server will assume that the application requires a Contact Center Express license to be issued.

AutoMonitorSplitOnAgentLogin

Syntax: *AutoMonitorSplitOnAgentLogin*

Description: When set (True), PhoneX will automatically monitor the split/skill device that an agent logs into.

Usage Notes

This is a Boolean property with a default value of True. When it is set to True, PhoneX will automatically monitor the particular split/skill an agent logs into.

CLIRestrictedReplacementString

Syntax: *CLIRestrictedReplacementString*

Description: When the calling party number received from the Telecom provider has presentation restriction, this string will replace the calling party number (*CallingDN*) in the call class.

Usage Notes

This property is a string property with default value set as “Restricted CLI”. This string will pass to the container application if the calling party number has a presentation restriction set. Change the property value if another value is required.

DisableSpecialDialSequence

Syntax: *DisableSpecialDialSequence*

Description: When set to True, special dial character intercepts will be disabled on a permanent basis.

Usage Notes

The property type is a Boolean with a default setting of False. This will remove any special dial character intercepts permanently.

HonorDefinityCLIRestriction

Syntax: *HonorDefinityCLIRestriction*

Description: When set to True and the calling party is restricted, PhoneX will replace the *CallerDN* number with the value found in *CLIRestrictedReplacementString*.

Usage Notes

This property is a Boolean property with a default value of True. If this property is set to True, the CLI will not display but rather a replacement string of the calling party number.

hWnd

Syntax:

Description:

Usage Notes

Index

Syntax: *Index As Integer*

Description: This property is used for creating a control array for PhoneX.

Usage Notes

Returns/sets the number identifying a PhoneX control in a control array.

IsConnected

Syntax:	*IsConnected*
Description:	This parameter is set when PhoneX connects to the Telephony Server.

Usage Notes

This property has default setting of False. When there is a connection to the Telephony Server, this property will be set to True.

MaximumCallAppearances

Syntax:	*MaximumCallAppearances*
Description:	A Read-only property of the maximum number of call appearances that are supported by PhoneX for a particular station device.

Usage Notes

A Read-only property that informs the user of the maximum number of call appearances able to be allocated for an individual station device.

MaxMonitoredDNs

Syntax:	MaxMonitoredDNs as long
Description:	Used to specify the maximum number of DN's that can be monitored by the PhoneX device.

Usage Notes

While present in the PhoneX COM interface this property is not currently used.

MaxOldCallListSize

Syntax:	*MaxOldCallListSize*
Description:	Sets the size of the old call list.

Usage Notes

This property has a default value of 100. The range for the history list of old calls is 10 to 1000. Newer calls will be placed at the bottom of the stack and the older calls will be removed once the limit has been exceeded.

MinimumCallAppearances

Syntax:	*MinimumCallAppearances*
Description:	A Read-only property of the minimum number of call appearances that are supported by PhoneX for a particular station device.

Usage Notes

A Read-only property that informs the user of the minimum number of call appearances able to be allocated for an individual station device.

Name

Syntax:

Description:

Usage Notes

Object

Syntax:

Description:

Usage Notes

Parent

Syntax:

Description:

Usage Notes

ReplaceUUIandCDwithOCIInfo

Syntax:

Description:

Usage Notes

i added this from aa manual.

A value that determines how the application handles user-to-user information (UUI) and collected digits (CD) when a call is transferred twice. If a call containing user information is transferred once, this information is transferred. But if the call is transferred a second time and the person making the transfer enters some new user information, *ReplaceUUIandCDwithOCIInfo* determines which set of user information is transferred. If the parameter is set to 0 (False), the application sends the new user information. If set to 1 (True), the application overwrites (replaces) the new information with the original user information (known as the original call information or OCI).

Tag

Syntax:

Description:

Usage Notes

TServers

Syntax:

Description:

Usage Notes

TraceActivity

Syntax:	*TraceActivity*
Description:	Sends PhoneX-related information to the parent container.

Usage Notes

This property has a default value of False. When set to True, PhoneX-related information will be sent to the parent container.

PollingSpeedAgentInfo

Syntax:	*PollingSpeedAgentInfo*
Description:	The speed at which PhoneX will poll the Definity switch/MultiVantage server to update the agent information.

Usage Notes

The default value for this property is 20, where the value specified is in seconds. The range for this property is 5 to 600. This is the frequency at which PhoneX will query the switch for agent information.

PollingSpeedFeatures

Syntax:	*PollingSpeedFeatures*
Description:	The speed at which PhoneX will poll the Definity switch/MultiVantage server to update the station features, eg. Send All Calls, Message Waiting and Call Forward.

Usage Notes

The default value for this property is 20, where the value specified is in seconds. The range for this property is 5 to 600. This is the frequency at which PhoneX will query the switch for Send All Calls, Call Forward and Message Waiting feature information.

PrivateVersion

Syntax:	*PrivateVersion*
Description:	The current private version data supported by the Telephony Server link.

Usage Notes

This property is only valid when there is an open telephony link. It will store the supported version of private data able to be used with the Telephony Server.

QueryACDStatus

Syntax:	*QueryACDStatus*
Description:	When set (True), PhoneX will query the Definity switch/MultiVantage server to update the status of all agents logged in.

Usage Notes

The default value for this property is True. When set, PhoneX will query the switch to update the status for all logged-in agents. The polling speed is set to that of *PollingSpeedAgentInfo*.

StripCLIRestrictionIndicator

Syntax:	*StripCLIRestrictionIndicator*
Description:	When set (True), PhoneX will remove the CLI Restriction indicator received with the calling party number from the Telecom provider.

Usage Notes

The property type is a Boolean with default setting of True. This will remove the CLI restriction placed by the provider on the calling party number.

TrunkIDReplacementString

Syntax:	*TrunkIDReplacementString*
Description:	Calls presented from the Definity switch/MultiVantage server that do not have a calling party number have the calling party field populated with a default string consisting of “T1#xxx”, where xxx represents the call ID. PhoneX will automatically replace this string with the value contained in *TrunkIDReplacementString*. If this field is zero length, the Definity/MultiVantage-provided string would be left unchanged.

Usage Notes

The default string value for this property is “Outside Call”. This will be the replacement string for any trunk calls. Change this property string if a different description is desired.

UpdateAgentStateOnCallClear

Syntax:	*UpdateAgentStateOnCallClear*
Description:	When set (True), PhoneX will automatically perform an update query on the Definity switch/MultiVantage server as to the agent state. This will ensure that automatic changes (eg. timed After Call Work) and manual logouts or state changes are reflected as soon as possible to the container application.

Usage Notes

This property is a Boolean property type. The default property setting is True. When set, it will cause PhoneX to automatically update the agent state by querying the switch on all logged-in agents.

CHAPTER 25

Appendix A - Special Dial Characters

PhoneX will accept the special dial characters in the *CalledDN* field. These special dial characters allow the user to manipulate the manner in which calls are dialed.

In This Chapter

Alphanumeric Characters

PhoneX converts alphanumeric characters into their equivalent numeric values:

Alphanumeric character	Numeric equivalent
ABC abc	2
DEF def	3
GHI ghi	4
JKL jkl	5
MNO mno	6
PQRS pqrs	7
TUV tuv	8
WXYZ wxyz	9

Example

If the *CalledDN* field in the *CallDial* method contains the string ‘1800Avaya’, PhoneX converts this to its numeric equivalent ‘180028292’.

Post-Dial DTMF

You can instruct PhoneX to send part of the dial string as in-band DTMF signalling. To do this, insert an exclamation mark (!) between the phone number and the post-dial digits.

Example

If the *CalledDN* field in the *CallDial* method contains the string '1800Avaya!8888#', PhoneX dials '180028292' and, when the call is answered, outpulses '8888#' as in-band DTMF signalling.

Note: To send DTMF digits from a Avaya Computer Telephony interface, you need a *StreamVersion* of 5. If the *CalledDN* contains an exclamation mark and the *StreamVersion* is '4', post-dial digits are ignored.

User-to-User Information

You can instruct PhoneX to include user-to-user information in the dial string. To do this, insert a semi-colon (;) between the phone number and the user-to-user information. The maximum length of user-to-user information currently accepted by the Definity ECS is 96 characters (assuming you have a Release 8 or better switch with Avaya Computer Telephony Release 3.30 Version 2.0 or higher on the Telephony Server; otherwise 32 characters for a switch prior to Release 8).

Note: The call class contains a variable specifically for user-to-user information. If this variable contains valid data (ie. it is not 0 length), the information gathered from the *CalledDN* variable is discarded.

Example

If the *CalledDN* field in the *CallDial* method contains the string ‘1800Avaya;Hello from Avaya’, PhoneX dials ‘180028292’ and sends ‘Hello from Avaya’ as user-to-user information.

User-to-User Information & Post-Dial DTMF Digits

You can also instruct PhoneX to send user-to-user information along with post-dial DTMF signalling. User-to-user information, however, must be included after the DTMF digits.

Note: Each special character sequence can only be included once, and any additional special dial characters are discarded.

Example

If the *CalledDN* field in the *CallDial* method contains the string, “1800Avaya!8888#;Hello from Avaya”, PhoneX dials ‘180028292’ and sends ‘Hello from Avaya’ as user-to-user information. When the call is answered, it outpulses ‘8888#’ as in-band DTMF digits.

If, however, the string was ‘1800Avaya;Hello from Avaya!8888#’, PhoneX dials ‘180028292’ and sends ‘Hello from Avaya!8888#’ as user-to-user information. It won’t recognize ‘8888#’ as post-dial DTMF digits.

CHAPTER 26

Appendix B - PhoneX Dial Control

For debugging purposes, information can be gained from PhoneX by issuing the following special dial sequences. These dial sequences will not be passed to the Telephony Server as a valid dial request.

In This Chapter

Version Numbers

If problems arise and you contact Avaya, the support personnel may ask you to supply version information. This can be gained by dialling the code sequence ‘0000000’ into PhoneX, ie. issuing a *CallDial* method with the *CalledDN* set to ‘0000000’. This will cause a series of dialog boxes to display with the version numbers of all the components being used by the Contact Center Express suite.

PhoneX Status Dump

You can instruct PhoneX to complete a status dump of all its internal parameters. The dump includes all call, device, agent and script classes, as well as the settings for all PhoneX properties.

The status dump can be started by dialling the code sequence ‘0000001’ into PhoneX, ie. issuing a *CallDial* method with the *CalledDN* set to ‘0000001’.

The status dump will be appended to the daily trace file. Tracing of PhoneX is stored in a text file located in the runtime directory of the container application. Trace files are automatically created with the name xxx_trc.txt, where xxx is the day of week (eg, Tue_trc.txt is Tuesday’s log file). Log files are created on a daily basis, giving a rolling, seven-day trace sequence. This trace dump will be completed regardless of whether PhoneX tracing is enabled or not.

PhoneX Tracing

In the customer release version of PhoneX, tracing is disabled. You can enable tracing from the host application by dialling the code sequence ‘0000002’ into PhoneX, ie. issuing a *CallDial* method with the *CalledDN* set to ‘0000002’. If tracing is enabled, you can disable it again by dialing the code sequence ‘0000003’ into PhoneX.

Tracing of PhoneX is stored in a text file located in the runtime directory of the container application. Trace files are automatically created with the name xxx_trc.txt, where xxx is the day of week (eg, Tue_trc.txt is Tuesday’s log file). Log files are created on a daily basis, giving a rolling, seven-day trace sequence.

Disable Special Dial Sequence

It is envisaged that the dial sequences mentioned on the previous page are not numbers required in the real world.

If this is not the case, you can disable the interception of these dial sequences from the host application by dialling the code sequence ‘0000099’ into PhoneX, ie. issuing a *CallDial* method with the *CalledDN* set to ‘0000099’. All special dial sequences will then be sent to the Telephony Server as normal make call requests.

Once disabled, you cannot re-enable the special dial intercept during the runtime life of the host application.

Note: An alternative way to disable the special dial character intercept is to set PhoneX’s *DisableSpecialDialSequence* property to True.

Index