Reject Cause 15

This feature, when enabled, will search ALL available RAT's if a Reject Cause 15 is sent by the network as the result of a cell selection. In the disabled state, the modem searches the same RAT from which it received the Reject Cause 15 message. Disabled is the normal search used by carriers other than CarrierX.

Name	SearchAllRatsAfterRejectCause15Enable
States	Enable, ***Disable***
Devices	U630, xU7x0, xU860/U870
FW Version	U630 – Build 21 and above xU7x0 – Build 30 and above xU860/U870 – Build 7 and above
AT Command	N/A
NV Item	NW_NV_SRCH_RATS_REJECT_15_I

N.A. CarrierZ

N.A. CarrierZ has mandated that Enhanced Network Selection (ENS) be enabled on all modems delivered to N.A. CarrierZ. This feature should also be enabled for all OEM customers launching on the N.A. CarrierZ network.

Enhanced Network Selection (ENS)

The N.A. CarrierZ network consists of different MNCs deployed in the various N.A. CarrierZ markets and the Empire and Triton markets having a non-N.A. CarrierZ MNC. GSM network selection procedures are an issue in that terminals perform their periodic higher priority PLMN scans even when on a N.A. CarrierZ network in spite of the display indicating a N.A. CarrierZ network. This causes customer issues when they travel from one N.A. CarrierZ market that uses one MNC to another N.A. CarrierZ market, which uses a different MNC. Additionally, Empire and Triton based customers are impacted more in that they are continuously in a roaming scenario on the T-Mobile/Triton network and yet from a billing, service and customer care perspective they are considered to be on the home N.A. CarrierZ network.

Name	EnsEnable
States	Enable, ***Disable***
Devices	xU7x0, xU860/U870
FW Version	All
AT Command	N/A
NV Item	NV_ENS_ENABLED_I

Factory Fulfillment

The following section details the features which relate to fulfillment of modems.

International Mobile Equipment Identity (IMEI)

The IMEI is a number unique to every GSM and UMTS mobile. It is found printed on the IMEI label located on the modem.

The IMEI number can be used by the GSM network to identify valid or network approved devices and is independent of the SIM.

The IMEI is a 15-digit number which includes information on the origin, model, and serial number of the device. The model and origin comprise the initial 8-digit portion of the IMEI, known as the Type Allocation Code (TAC). The remainder of the IMEI is manufacturer-defined, with a Luhn check digit at the end (which is never transmitted).

As of 2004, the format of the IMEI is as follows: *AA-BBBBBB-CCCCCC-D*

where: *AA* – Reporting Body Identifier, indicating the GSMA – approved group that allocated the modem TAC code BBBBB,
BBBBBB – TAC
CCCCCC – serial number of the modem model
D – Luhn check digit for the entire number

Name	IMEI
States	OEM customers will supply range corresponding to their GCF or PTCRB grant. NVTL has its own IMEI for generic cards.
Devices	U630, xU7x0, xU860/U870
FW Version	All
AT Command	AT+CGSN
NV Item	NV_UE_IMEI_I

E725 Provisioning with IOTA

This applies only to the E725. The EU860D & EU870D use SIM cards and don't require any type of IOTA.

Sprint PCS uses IOTA to perform their provisioning before a wireless device is allowed on the data network. This process is operator specific so there maybe variations as to how provisioning is done. In all cases, please contact the network operator if you have questions concerning activation and subscriber related questions.

When using the PCI Express Mini Card, the activation is done by MobiLink™. MobiLink™ will automatically detect if the E725 module needs to perform any provisioning on Sprint's network.

If an E725 module does not use MobiLink, you must run IOTA from the primary port of the E725 module. Novatel Wireless has developed an embedded IOTA Client called, eIOTA that interfaces through AT commands. This Client will allow the subscriber to execute an IOTA session to provision the E725. Once this is done, the E725 can access the operator's 1xRTT and 1xEVDO networks.

For use with Sprint PCS, the subscriber first needs to contact a sales representative to activate the E725. The Sprint PCS representative will present to the subscriber the MDN or MIN numbers with the SPC. These parameters need to be entered into the E725 if it does not already exist. Upon the time of receiving these parameters, Sprint PCS has a time provisioning requirement of 1.5 days to 2 days for the E725 to perform and complete an IOTA session. If the subscriber does not complete the IOTA provisioning within this time, the subscriber will have to call Sprint PCS again to reset the provisioning timer.

eIOTA

eIOTA is a subscriber unit provisioning Client, or Provisioning Service Agent. Embedded in the CDMA wireless modem, the Client communicates with Handset Configuration Manager, the operator's IOTA server, to download provisioning data to the subscriber unit or upload settings per server's request. It allows the operator to remotely perform provisioning without having to bring the wireless device into a sales location.

eIOTA is disabled by default from the factory. This is done because if eIOTA was active, it would automatically attempt an eIOTA session if the E725 has not already completed provisioning. When the subscriber finishes entering the MDN or MIN, they could either enable eIOTA and have the E725 automatically attempt an IOTA session after a power cycle or initiate a manual IOTA session.

Enabling, disabling, and starting eIOTA

eIOTA Client can be enabled or disabled by issuing the AT commands:

To enable: AT+IOTA=1

To disable: AT+IOTA=0

To force start: AT+IOTA=2

There are two ways to start eIOTA, NIIP(Network Initiated Initial Provisioning) or CIIP(Client Initiated Initial Provisioning). In NIIP, operator's IOTA server pushes a special SMS message to the Client to trigger an IOTA session. In CIIP, a session can be triggered by locally issuing an AT command: AT+IOTA=2.

Checking eIOTA status

The AT command: AT+IOTA=? Is used to query the eIOTA status while IOTA is active.

Please refer to AT+IOTA in the AT Commands Chapter for details.

Cautions that need to be taken when eIOTA is active

DO NOT power off the unit until IOTA session is finished.

DO NOT remove the antenna from the unit.

DO NOT disconnect the data call issued by eIOTA.

When running eIOTA, to ensure no power lost, make sure to use the AC power and NOT the battery power.

Appendix B - Development Tools & Procedures

Introduction

This appendix contains identification and introductory guidance on development tools used in the validation, certification and compliance testing of the Novatel Wireless PCI Express Mini-cards

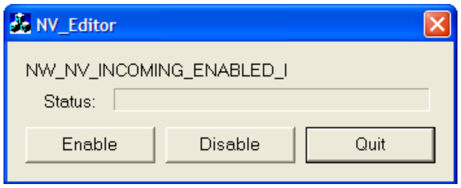
Satimo Chamber

Novatel Wireless has the in-house ability* to test TRP/TIS on host devices integrating the Novatel Wireless PCI Express Mini card. It is of great value to pre-scan the integrated device for both development and meeting carrier TRP/TIS requirements.   Please contact Novatel Wireless for further information.

* Sept 2006

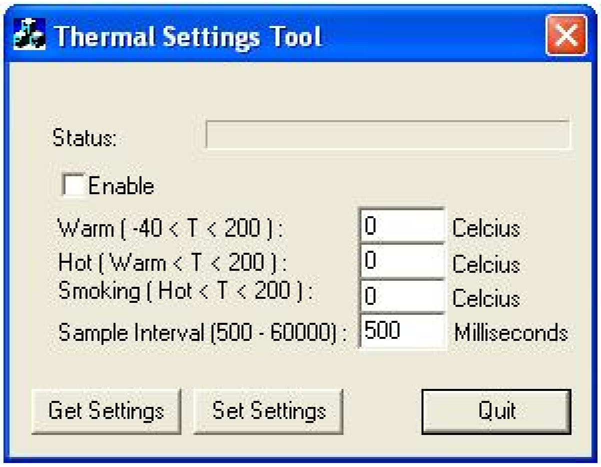
HSDPA: Set Mini card to Circuit Switch (CS) Mode

In order to assist in the execution of FCC/CE testing Novatel Wireless has developed an executable that will force the EU860D/EU870D into CS mode of operation. This tool allows for configuring a manual mobile originated/terminated GSM circuit switched voice call on the Novatel Wireless EU8X0 modem in a test lab environment. Please contact Novatel Wireless for further information.



Thermal Limit Tool

The Thermal Limit Tool is used for the purpose of enabling/disabling, or modifying, the thermal shutdown settings of the EU8X0 modem.



Status:

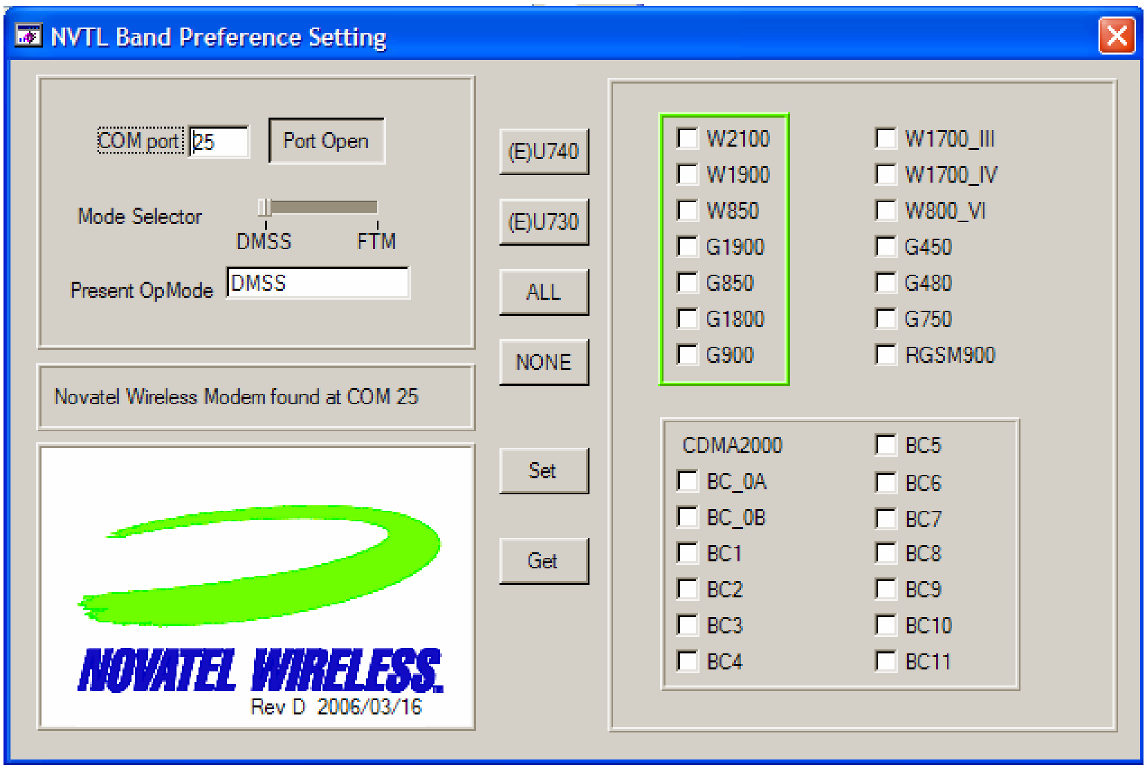
□ Enable

Warm (-40 < T < 200) : 0 Celcius
Hot (Warm < T < 200) : 0 Celcius
Smoking (Hot < T < 200) : 0 Celcius
Sample Interval (500 - 60000) : 500 Milliseconds

Band Preference Tool

The band preference tool enables or disables specific bands on the Novatel Wireless EU8X0 modem in a test lab environment.

When executing conducted RF tests the PCI Express Mini card can also receive over the air signaling from commercial networks. The PCI Express Mini card may try to connect to these commercial networks (depending on the modem configuration) and not connect to the intended conducted band. This tool allows the disabling of specific bands.



NVTL Band Preference Setting

COM port: 25

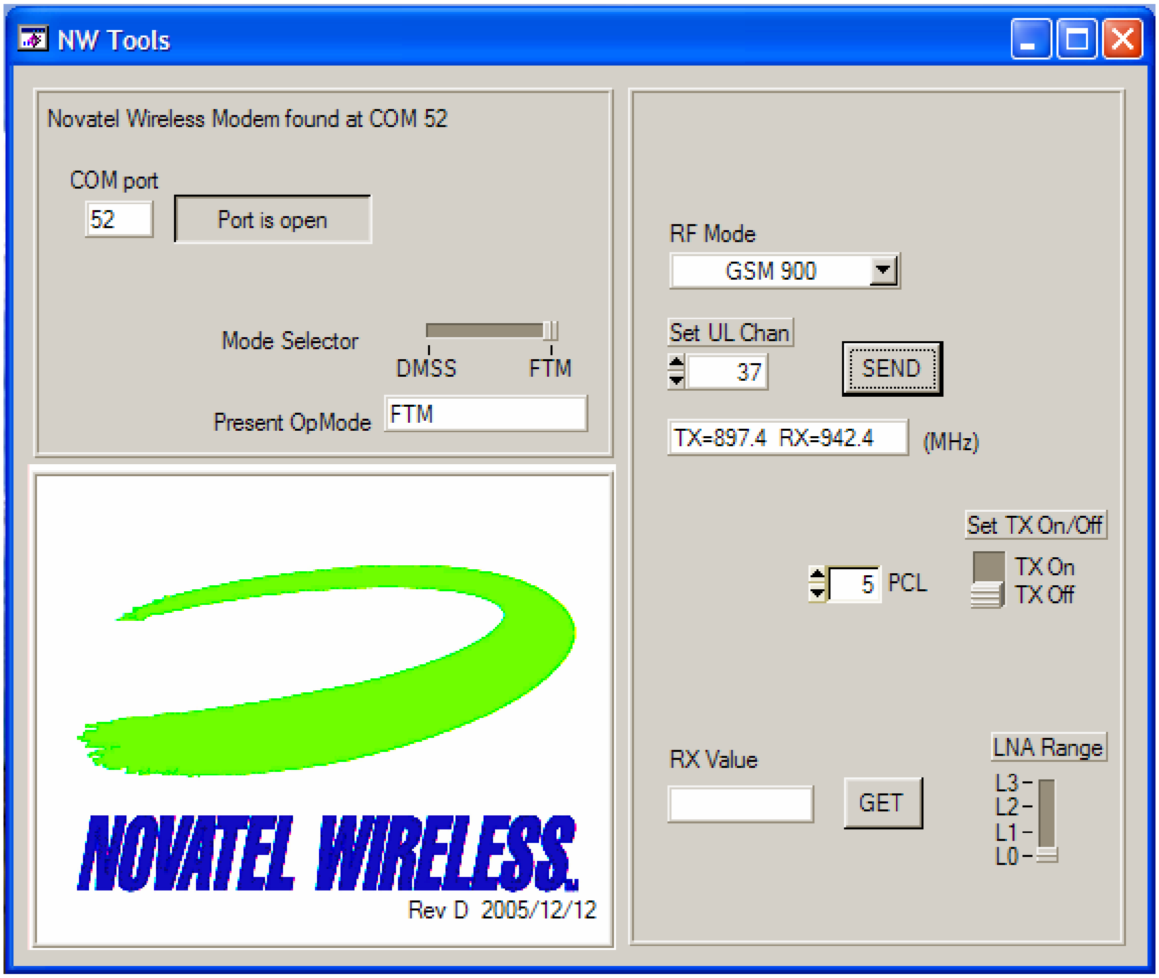
Present OpMode: DMSS

Novatel Wireless Modem found at COM 25

Band Preference Application Tool Box

HSDPA: RF Application

The RF Application and procedure provides the instructions of how to set up and use the Novatel Wireless RF application tool for the purpose of performing TX and RX lab environment tests on the Novatel Wireless EU8X0 HSDPA modem.



Novatel Wireless Modem found at COM 52

COM port: 52

Port is open

Present OpMode: FTM

RF Mode: GSM 900

Set UL Chan: 37

TX=897.4 RX=942.4 (MHz)

5 PCL

Figure 1 – NW RF Application Tool Box

EVDO RF Application

In order to assist in the execution of FCC testing Novatel Wireless has developed a test setup application that is used to configure the E725 for FCC testing. Please contact Novatel Wireless for father information.

Appendix C - Regulatory Approval and Compliance

As both the EU860D and EU870D support four bands of GPRS operation, including North American and European bands, both products are covered by regulatory requirements of North America and Europe. Both products will have FCC, PTCRB, CE and GCF certification.

The E725, as a CDMA product in North America requires FCC certification.

FCC (Federal Communication Commission)

The E725, EU870D and EU860D products conform to the requirements of applicable American laws with respect to safety; health, environment and consumer protection.

This E725, EU870D and EU860D will comply, per applicable band, with the following parts of the Federal Communication Commission's (FCC) Code of Federal Regulations (CFR):

FCC CFR47 Part 2 (General Rules and Regulations, RF Exposure Evaluation)

FCC CFR47 Part 15 (All Radio Frequency Devices)

FCC CFR47 Part 24 (Narrow and wideband PCS modules)

FCC CFR47 Part 22 (Cellular Service)

A FCC grant shall be obtained in order to demonstrate compliance.

CE (Conformance European)

The EU870D and EU860D products comply with the essential requirements of the applicable European laws and directives with respect to safety; health, environment and consumer protection. The products conform to the essential requirements of the R&TTE (Radio and Telecommunications Terminal Equipment) Directive, 1999/5/EC, and have the CE mark affixed. The applicable sections of the following standards have been used to demonstrate compliance to this requirement. The EU860D and EU870D products will comply with the 3GPP standards TS 51.010 for GSM and TS 34.121 for WCDMA.

R&TTE

R&TTE Requirement	Discipline	Definition	Applied Standard
Article 3.1(a)	Health	Safety Testing (flammability, etc…)	ICNIRP 1998[5] European Council Rec.1999/519 EC
Article 3.1(a)	Safety		IEC 60950-1[6]
Article 3.1(b)	EMC	EMC testing (unintentional radiators, etc.…)	EN 301 489-01[7]
			EN 301 489-07[8]
			EN 301 489-24[9]
Article 3.2	Spectrum	Network Testing (power, frequency stability, etc…)	EN 301 511[10]
			EN 301 908-1[11]
			EN 301 908-2[12]

[5] International Commission on Non-Ionizing Radiation Protection

[6] Safety of Information Technology Equipment

[7] Electromagnetic compatibility and Radio Spectrum Matters (ERM) ElectroMagnetic Compatibility (EMC) standard for radio equipment and services
Part 1: Common Technical requirements

[8] Electromagnetic compatibility and Radio Spectrum Matters (ERM) ElectroMagnetic Compatibility (EMC) standard for radio equipment and services
Part 7: Specific conditions for mobile and portable radio and ancillary equipment of digital cellular radio telecommunications systems (GSM and DCS)

[9] Electromagnetic compatibility and Radio Spectrum Matters (ERM) ElectroMagnetic Compatibility (EMC) standard for radio equipment and services
Part 24: Specific conditions for IMT-2000 CDMA Direct Spread (URTA) for Mobile and portable radio and ancillary equipment.

[10] Global System for Mobile communications (GSM):
Harmonized EN for mobile stations in the GSM 900 and GSM1800 bands covering essential requirements under article 3.2 of the R&TTE directive

[11] Electromagnetic compatibility and Radio Spectrum Matters (ERM) Base Stations (BS) and User Equipment (UE) for IMT-2000 Third-Generation cellular networks.

[12] Electromagnetic compatibility and Radio Spectrum Matters (ERM) Base Stations (BS) and User Equipment (UE) for IMT-2000 Third-Generation cellular networks.
Part 2: Harmonized EN for IMT-2000,

The EU860D and EU870D products will comply with the applicable GSM/GPRS European Regional Regulatory Requirements as per the following table.

GSM/GPRS European Regulations

GSM 11.10 / TS 51.010 Requirement	***Description***	***EU / R&TTE Directive***
12.1.1	Conducted spurious emissions - MS allocated a channel	Yes
12.1.2	Conducted spurious emissions - MS in idle mode	Yes
12.2.1	Radiated spurious emissions - MS allocated a channel	Yes
12.2.2	Radiated spurious emissions - MS in idle mode	Yes
13.1	Transmitter – Frequency error and phase error	Yes
13.2	Transmitter – Frequency error under multipath and interference conditions	Yes
13.3-1	Transmitter output power and burst timing - MS with permanent antenna connector	Yes
13.4	Transmitter - Output RF spectrum	Yes
13.6	Transmitter – Frequency error and phase error in HSCSD multislot configuration	Yes
13.7	Transmitter output power and burst timing in HSCSD configurations	Yes
13.8	Transmitter, Output RF spectrum in HSCSD multislot configuration	Yes
13.16.1	Frequency error and phase error in GPRS multislot configuration	Yes
13.16.2	Transmitter output power in GPRS multislot configuration	Yes
13.16.3	Output RF spectrum in GPRS multislot configuration	Yes
13.17.1	Frequency error and Modulation accuracy in EGPRS Configuration	Yes
13.17.2	Frequency error under multipath and interference conditions in EGPRS Configuration	Yes
13.17.3-1	EGPRS Transmitter output power- MS with permanent antenna connector	Yes
13.17.4	Output RF spectrum in EGPRS Configuration	Yes
14.7.1	Blocking and spurious response - speech channels	Yes
14.18.5	Blocking and spurious response in EGPRS Configuration	Yes

CDMA Direct Spread (UTRA FDD) (UE) covering essential requirements of article 3.2 of the R&TTE Directive

GCF (Global Certification Forum)

The EU870D product will be tested to and meet the GCF CC (Certification Criteria) requirements in order to comply with Regional Regulatory Requirements.

Novatel Wireless will provide a full GCF declaration for the EU870D including GCF-AP Annex C, D, E and F based on GCF-CC.

GCF Version Compliance[13]

Commercial Release

- Release 4.0E Patch 3 GCF 3.22

Post Commercial Upgrade

- Release 4.0E Patch 4 GCF 3.23

[13] Targeted Firmware release and GCF Compliance versions are subject to change without notice. Please contact Novatel Wireless for actual Firmware release and GCF compliance version

PTCRB (PCS Type Certification Review Board)

The EU860D product will be compliant to PTCRB.

PTCRB Version Compliance[14]

Commercial Release

• Release 4.0E Patch 3	PTCRB 3.7.1

Post Commercial Upgrade

• Release 4.0E Patch 4	PTCRB 3.9.1

[14] Targeted Firmware release and PTCRB Compliance versions are subject to change without notice. Please contact Novatel Wireless for actual Firmware release and PTCRB compliance version

Appendix D - Carrier Accreditation and Infrastructure IOT

Carrier Accreditation

Contact Novatel Wireless for further information and details on Carrier Accreditation.

Infrastructure IOT

Novatel Wireless works with the carriers and infrastructure vendors to complete Interoperability testing with the PCI Express Mini Card in either lead host devices or in stand alone testing. Contact Novatel Wireless for further information and details on Carrier Accreditation.

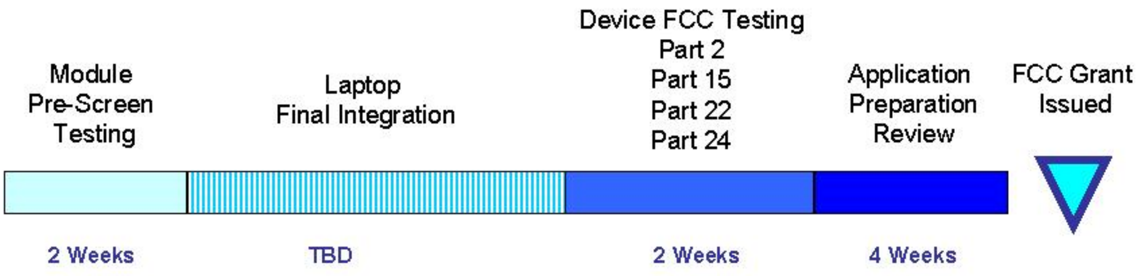
.

Estimated Timelines for Compliance & Certification

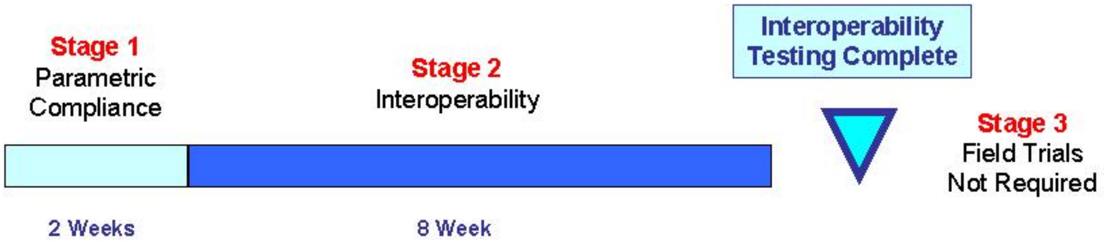
Disclaimer

The Estimated Timelines is provided for informational purposes only. Novatel Wireless is providing the Information because it may be useful. The Information is provided solely on the basis that you will be responsible for making your own assessments of the information and are advised to verify all representations, statements and information before using or relying upon any of the Information. Although Novatel Wireless believes it has exercised reasonable care in providing the Information, Novatel Wireless does not warrant the accuracy of the information and is not responsible for any damages arising from the use of or reliance upon the information. Novatel Wireless in no way represents that Novatel Wireless is providing the information in accordance with any standard or service (routine, customary or otherwise) related to the consulting, services, hardware or software industries.

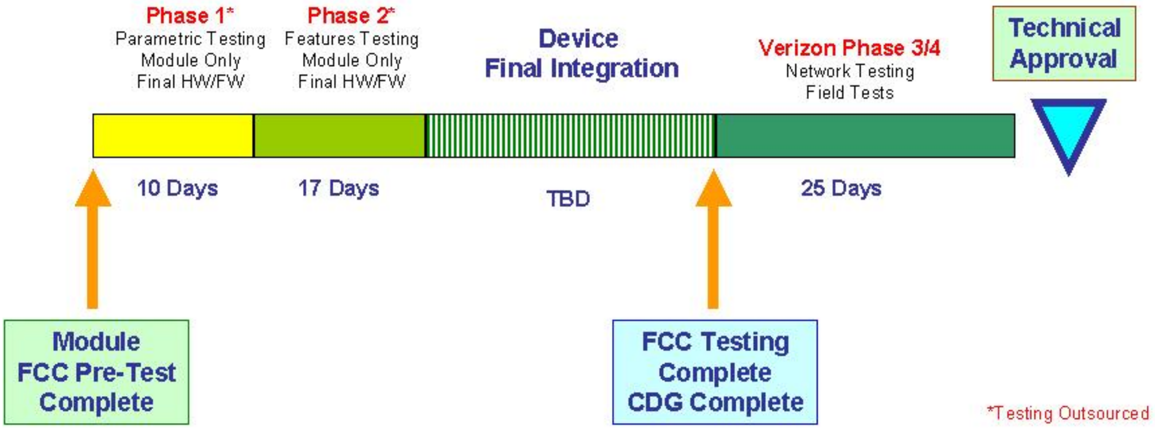
EV-DO FCC Accreditation


Module Pre-Screen Testing
Laptop Final Integration
Device FCC Testing Part 2 Part 15 Part 22 Part 24
Application Preparation Review
FCC Grant Issued
2 Weeks
TBD
2 Weeks
4 Weeks

EV-DO CDG Interoperability

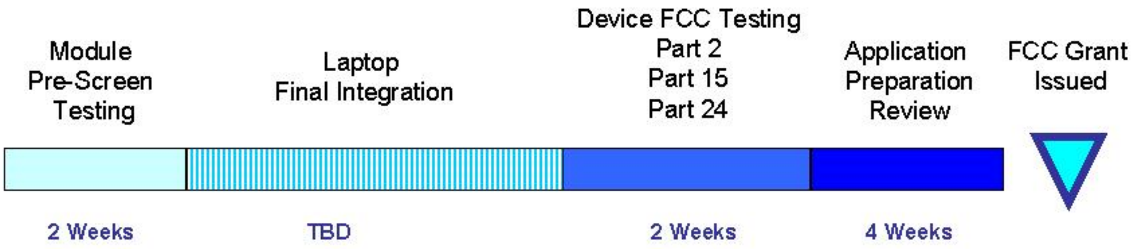

Stage 1 Parametric Compliance
Stage 2 Interoperability
Interoperability Testing Complete
Stage 3 Field Trials Not Required
2 Weeks
8 Week

EV-DO Verizon Certification Process



HDSPA FCC Accreditation

HDSPA FCC Accreditation takes approximately 4 weeks until an FCC grant is issued.



Notes:

Hardware should be near final, any changes after a module pre-screen are handled as either a Class I or Class II permissive change, depending on their scope.

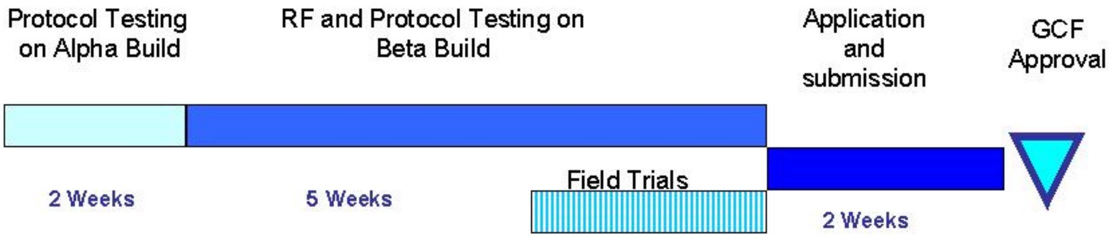
GCF Compliance Process

The EU870D product will be tested for compliance to GCF as per the applicable GCF test criterion at the time of testing. Novatel Wireless is a current member of the GCF (Global Certification Forum). GCF quarterly meetings are attended in order to keep appraised of new procedures, policies and technical requirements associated with GCF terminal certification.

GCF certification is very powerful. Because it provides for parametric as well as protocol and field test plans, certification under this body can be highly leveraged to gain accreditation on carrier networks. For North American, PTCRB provides similar coverage to GCF. As such, results from either GCF or PTCRB can be leveraged to one another.

Typically 9 weeks until GCF Approval.

Spurious Emissions is required on the host laptop/handheld to achieve compliance on the integrated system.


Protocol Testing on Alpha Build
RF and Protocol Testing on Beta Build
Application and submission
GCF Approval
2 Weeks
5 Weeks
Field Trials
2 Weeks

Notes:

GCF provides a framework to have devices accepted by most carriers in Europe.

There is a large amount of overlap with PTCRB (North American equivalent). Results from PTCRB can be leveraged for GCF and vice versa.

Unlike PTCRB, this is voluntary. However, it is required by Vodafone.

It includes RF performance, emissions, protocol and field performance test cases. It is a kin to the CDG Stage 1, Stage 2, & Stage 3 recommendations for CDMA.

Requires mandatory testing in at least 5 networks.

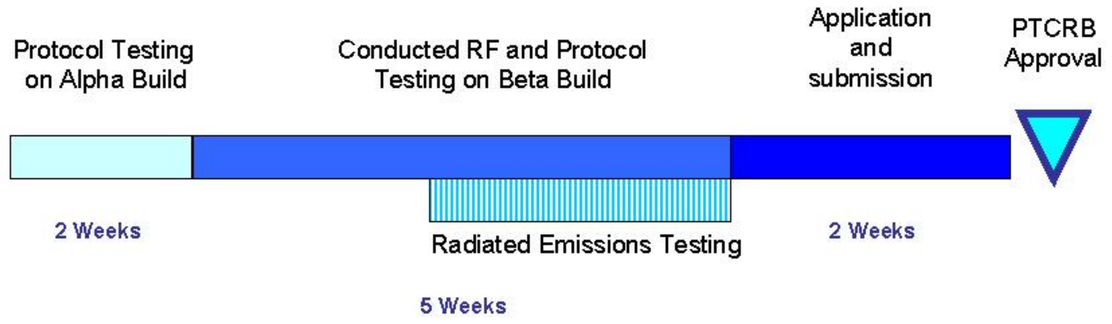
PTCRB Compliance Process

The EU860D product will be tested for compliance to PTCRB as per the applicable PTCRB test criterion at the time of testing.

North America – PTCRB (PCS Type Certification Review Board)

Novatel Wireless is a current Member of the PTCRB (PCS Type Certification Review Board). PTCRB quarterly meetings are attended in order to keep appraised of new policies, procedures and technical requirements associated with GCF terminal certification. Novatel Wireless has attained PTCRB approval with several product offerings.

Spurious Emissions & SIM electrical is required on the host laptop/handheld to achieve compliance on the integrated system.


Protocol Testing on Alpha Build
Conducted RF and Protocol Testing on Beta Build
Application and submission
PTCRB Approval
2 Weeks
Radiated Emissions Testing
2 Weeks
5 Weeks

Notes:

North American equivalent of GCF.

This a group comprised of both Operators and Manufacturers.

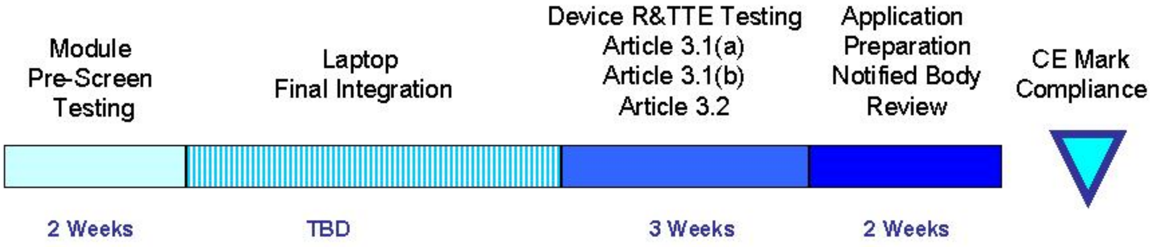
The radiated emission testing is similar to the FCC requirements but not entirely identical.

This is mandatory in North America to operate GSM devices in the PCS band. There are approximately 1,200 test cases.

Any changes (MMI, RF, Baseband, etc…), no matter how insignificant which occur after certification, are to be reported to the PTCRB for review.

CE Mark Certification Process

The CE Mark Certification process takes approximately 7 weeks until CE Mark Compliance is complete.



Notes:

Module hardware should be final at module pre-screen. However, modifications are self policed and are less restrictive than FCC permissive change policies.

R&TTE governs the CE initiative.

Article Definition

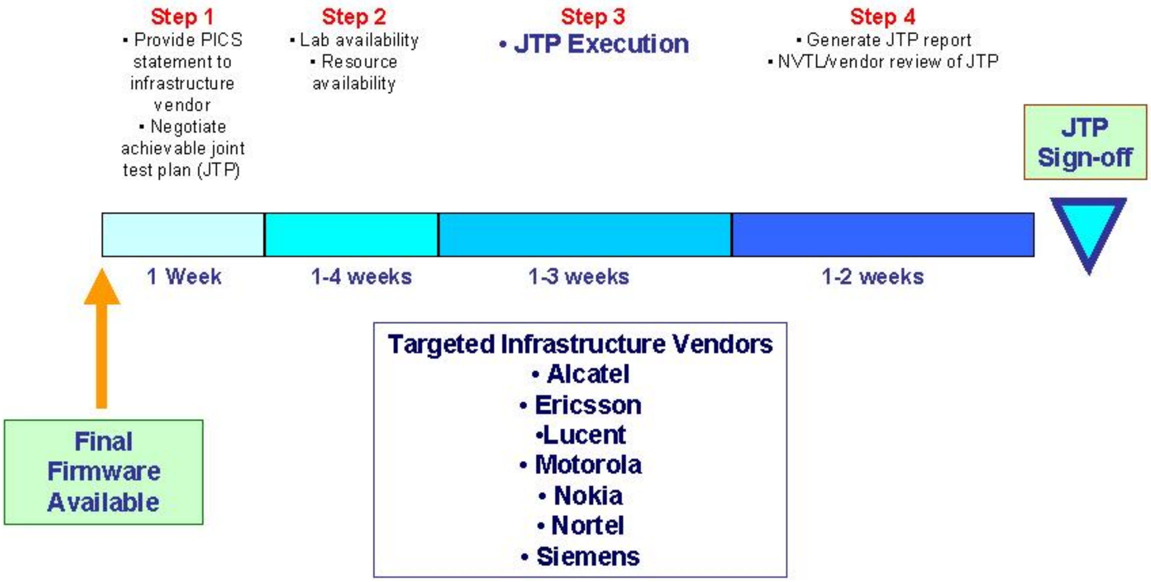
Article 3.1 (a) – Safety Testing (flammability, etc…)

Article 3.1 (b) – EMC testing (unintentional radiators, etc.…)

Article 3.2 – Network Testing (power, frequency stability, etc…)

Infrastructure IOT Process

A Protocol Implementation Control Statement (PICS) statement will be provided outlining the protocol supported in Qualcomm™'s stack as integrated into EU860D and EU870D code release. This will be used to plan IOT test cases.



Carrier Accreditation Process

The typical Carrier Accreditation roadmap is shown in the two example diagrams ***N.A. HSDPA Carrier Accreditation*** and ***European HSDPA Carrier Accreditation.***

Novatel Wireless works with the carriers and infrastructure vendors to complete Accreditation Testing with the PCI Express Mini Card in either a lead device (such as a PC card or Express card), a lead host laptop or in stand alone testing.

With the mini card or PC card validated on the network the carrier looks to the ***host laptop/handheld integrator to complete incremental GCF or PTCRB testing on the*** laptop/handheld.

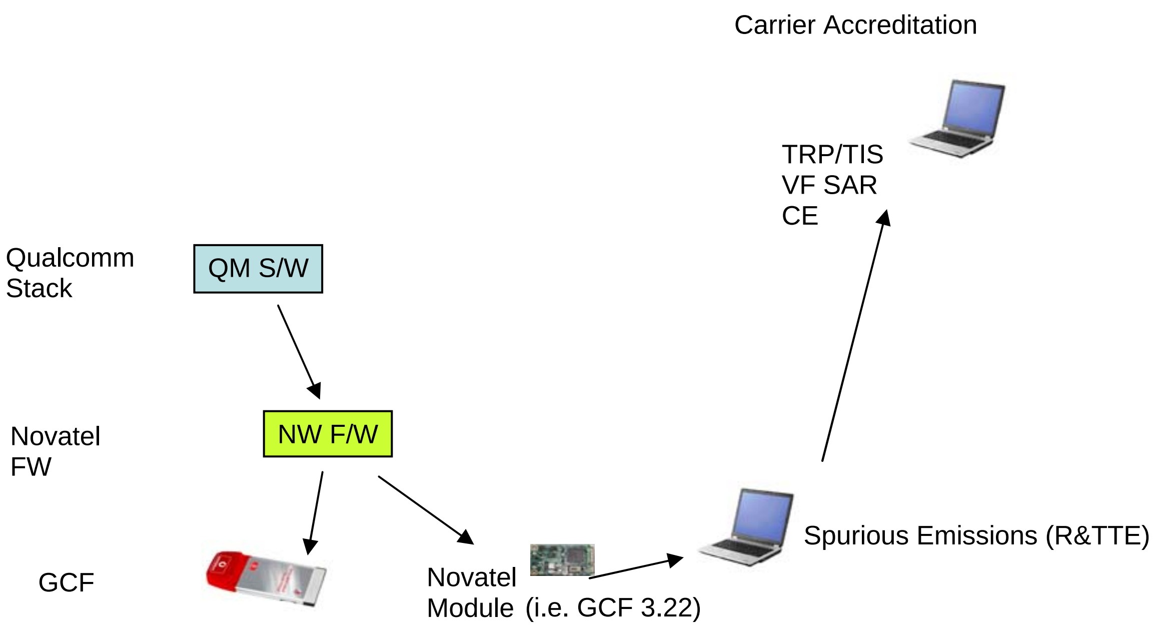
Novatel attains GCF or PTCRB on the core PCI Express Mini-card which is the bulk of the testing and test cases. ***Typically the incremental GCF or PTCRB testing on the integrated system consists of SIM Electrical*** [15] ***and Spurious Emissions***.

Host integrator completes TRP/TIS measurements. These results are required to be compliant to carrier's TRP/TIS requirements. In some cases the carrier will have additional requirements specific to the carrier ***such as incremental SAR testing***.

The Carrier will require several laptops/handhelds configured with the PCI Express Mini Card to evaluate in additional to requiring PTCRB/GCF compliance and TRP/TIS test results.

Contact Novatel Wireless for further information and details on carrier accreditation.

Europe HSDPA Carrier Accreditation



Carrier Accreditation

TRP/TIS
VF SAR
CE

Qualcomm Stack

QM S/W

Novatel FW

NW F/W

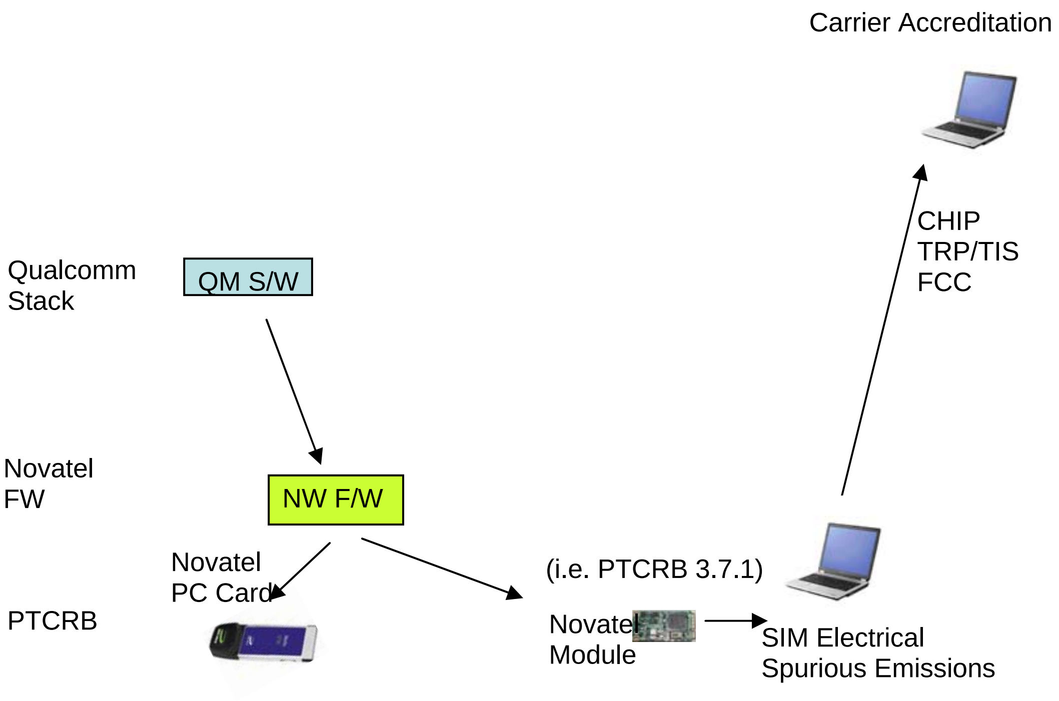
GCF

Novatel Module (i.e. GCF 3.22)

Spurious Emissions (R&TTE)

[15] SIM Electrical testing optional for GCF

N.A. HSDPA Carrier Accreditation



Carrier Accreditation

CHIP
TRP/TIS
FCC

Qualcomm
Stack

QM S/W

Novatel
FW

NW F/W

Novatel
PC Card

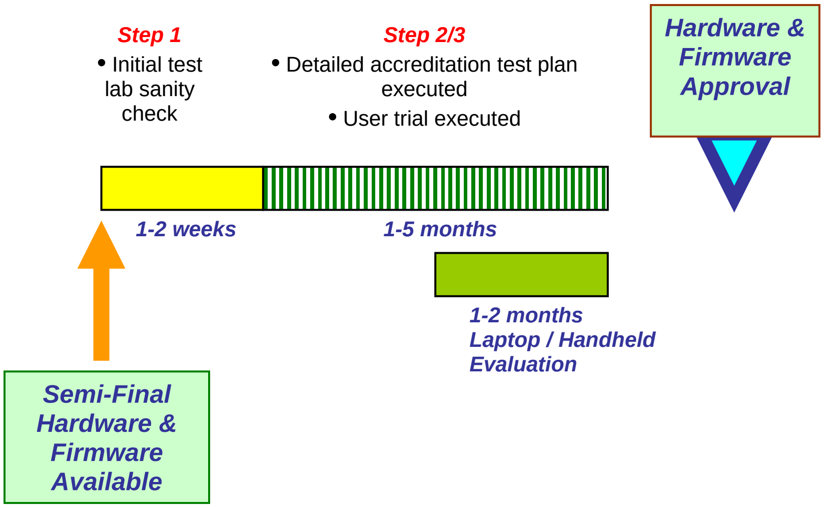
PTCRB

(i.e. PTCRB 3.7.1)

Novatel
Module

SIM Electrical
Spurious Emissions

Carrier Certification Process



Step 1

- Initial test lab sanity check

Step 2/3

- Detailed accreditation test plan executed
- User trial executed

Hardware & Firmware Approval

1-2 weeks

1-5 months

1-2 months
Laptop / Handheld
Evaluation

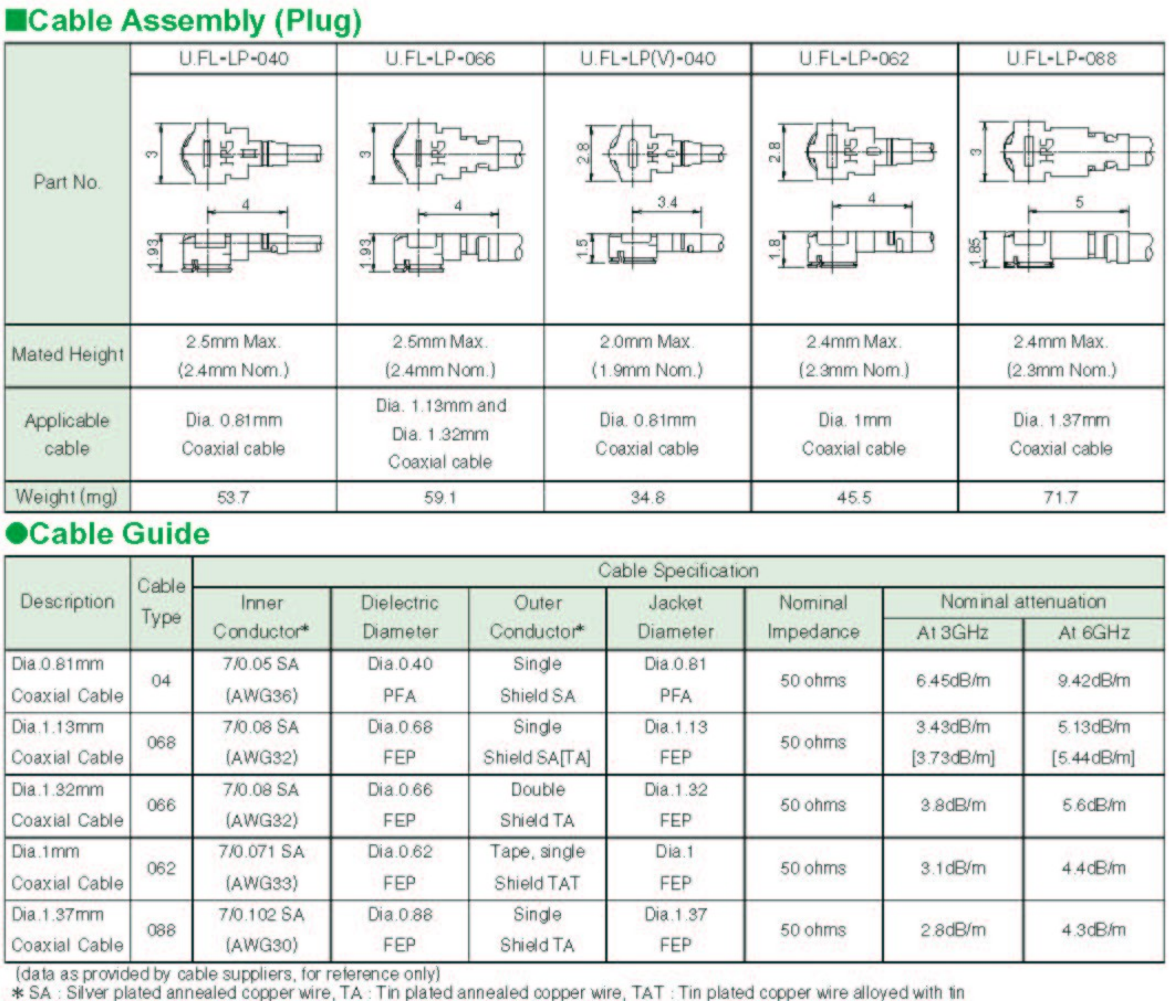
Semi-Final Hardware & Firmware Available

Appendix E - Reference Parts Specifications

RF Connector

Hirose U.FL series, with U.FL-R-SMT receptacle mating to the following connectors illustrated. Note that . This connector is designed for a limited number of insertions. For an embedded application this is expected to be acceptable.

Figure 49: RF Connector



3
4
1.93
HRS
3
4
1.93
2.8
3.4
1.5
2.8
4
1.8
3
5
1.85

■Cable Assembly (Plug)

Part No.	U.FL-LP-040	U.FL-LP-066	U.FL-LP(V)-040	U.FL-LP-062	U.FL-LP-088
Mated Height	2.5mm Max. (2.4mm Nom.)	2.5mm Max. (2.4mm Nom.)	2.0mm Max. (1.9mm Nom.)	2.4mm Max. (2.3mm Nom.)	2.4mm Max. (2.3mm Nom.)
Applicable cable	Dia. 0.81mm Coaxial cable	Dia. 1.13mm and Dia. 1.32mm Coaxial cable	Dia. 0.81mm Coaxial cable	Dia. 1mm Coaxial cable	Dia. 1.37mm Coaxial cable
Weight (mg)	53.7	59.1	34.8	45.5	71.7

●Cable Guide

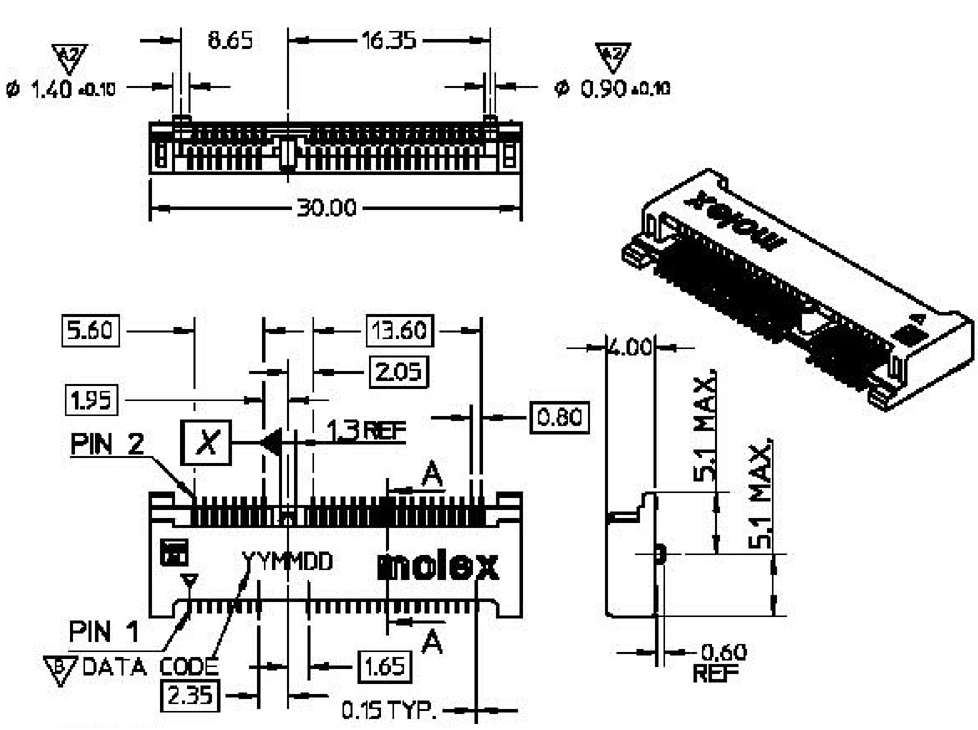
Description	Cable Type	Cable Specification						
		Inner Conductor*	Dielectric Diameter	Outer Conductor*	Jacket Diameter	Nominal Impedance	Nominal attenuation	
							At 3GHz	At 6GHz
Dia.0.81mm Coaxial Cable	04	7/0.05 SA (AWG36)	Dia.0.40 PFA	Single Shield SA	Dia.0.81 PFA	50 ohms	6.45dB/m	9.42dB/m
Dia.1.13mm Coaxial Cable	068	7/0.08 SA (AWG32)	Dia.0.68 FEP	Single Shield SA[TA]	Dia.1.13 FEP	50 ohms	3.43dB/m [3.73dB/m]	5.13dB/m [5.44dB/m]
Dia.1.32mm Coaxial Cable	066	7/0.08 SA (AWG32)	Dia.0.66 FEP	Double Shield TA	Dia.1.32 FEP	50 ohms	3.8dB/m	5.6dB/m
Dia.1mm Coaxial Cable	062	7/0.071 SA (AWG33)	Dia.0.62 FEP	Tape, single Shield TAT	Dia.1 FEP	50 ohms	3.1dB/m	4.4dB/m
Dia.1.37mm Coaxial Cable	088	7/0.102 SA (AWG30)	Dia.0.88 FEP	Single Shield TA	Dia.1.37 FEP	50 ohms	2.8dB/m	4.3dB/m

(data as provided by cable suppliers, for reference only)
∗ SA : Silver plated annealed copper wire, TA : Tin plated annealed copper wire, TAT : Tin plated copper wire alloyed with tin

Mini Card Connector

Molex 67910 series, mates with the mini PCI Express Card. Use with Latch 48099-0003.

Figure 50: Mini PCI Express Connector



Appendix G - Phoenix API Interface to PCI Express Mini Card

Overview

This is the Phoenix API Command Set Reference for the Novatel Wireless CDMA Modem product. This document describes the modem API used by host applications running on Windows 2000, and Windows XP.

The SDK provides universal API support for both 1XEV-DO and HSDPA mini-cards. This provides interfaces through the Windows XP and Windows 2000 operating systems. It includes API support as well as sample code to provide for ease of application development.

Phoenix is the brains of the SDK. Phoenix maintains a single state machine that all Clients communicate with. Anything and everything involving communication to the device takes places through the Phoenix server. Implemented as a Document/View executable supporting automation, the Phoenix server automatically keeps a count of how many Clients are attached to it via COM interfacing. The server is initialized automatically once the first Client is instantiated and shut down once the last Client instance is terminated. With the beauty of OLE Automation, the Phoenix server can be utilized using many different programming languages, including C++, MFC, JavaScript, VBScript, etc. Refer to Phoenix.chm for API documentation. If you want to use Phoenix in Visual Studio, import the type library Phoenix.tlb and create a wrapper class for it.

Blaze ActiveX control helps Client applications to receive events fired by the Phoenix server. This allows for simple 2-way communication, replacing redundant loop checking used in the past. Refer to Blaze.chm for API documentation. If you want to use Blaze ActiveX control in Visual Studio, add the NVTL Blaze control from the registered Components and Controls Gallery and create a wrapper class for it.

Phoenix API is the communication engine between host applications and a Novatel Wireless CDMA modem. It is a DLL library that provides an interface for user/host applications to communicate commands to the modem for purposes of serial access, general diagnostic, NV programming, SMS messaging, and general modem functions. It provides the hardware abstraction that the host applications don’t need to involve itself with.

The following facts and conventions are applicable across the whole document unless specially specified.

All API calls are synchronous. The calling thread will be blocked until the function call returns.

HANDLE hCom is used in most Loader functions as the first argument. It will not be repeated in the Parameters section for every function. The com port handle must be obtained by calling function Open_Output_Handles( ). NULL is not a valid handle value. The handle should be closed before applications quit.

Modem: Novatel Wireless CDMA Modem

Loader: Novatel Wireless CDMA Modem Loader API

Applications: Host applications using Loader API to access Novatel Wireless CDMA PC Card Modem

Client Object

The Client object uses the following methods:

ChangeLockCode method

Description: Used to change the lock code of the device.

Return Type: A ***Long*** value.

Syntax: object***.ChangeLockCode***(lpszLockCode As String, lpszNewLockCode As String)

The ChangeLockCode method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
lpszLockCode	String
lpszNewLockCode	String

Sample Code using ChangeLockCode Method:

```
long IPhoenixWrapper::ChangeLockCode(LPCTSTR lpszLockCode, LPCTSTR
lpszNewLockCode)
{
        long result;
        static BYTE parms[] =
                VTS_BSTR VTS_BSTR;
        InvokeHelper(0x18, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                lpszLockCode, lpszNewLockCode);
        return result;
}
```

Connect method

Description: To initiate a PS or CS (if supported by device) call.

Return Type: ***A Long value.***

Syntax:
*object**.Connect**(lpszUsername As String, lpszPassword As String, lpszNumber As String, varErrorMsg As Variant, nIPAddress As Long, nPrimaryDNS As Long, nSecondaryDNS As Long, nPrimaryWINS As Long, nSecondaryWINS As Long, nPapChap As Long, lpszVPN As String*)

The Connect method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
lpszUsername	String
lpszPassword	String
lpszNumber	String
varErrorMsg	Variant
nIPAddress	Long
nPrimaryDNS	Long
nSecondaryDNS	Long
nPrimaryWINS	Long
nSecondaryWINS	Long
nPapChap	Long

lpszVPN	String

Sample Code using Connect Method:

```
long IPhoenixWrapper::Connect(LPCTSTR lpszUsername, LPCTSTR lpszPassword, LPCTSTR
lpszNumber, VARIANT* varErrorMsg, long nIPAddress, long nPrimaryDNS, long
nSecondaryDNS, long nPrimaryWINS, long nSecondaryWINS, long nPapChap, LPCTSTR
lpszVPN)
{
        long result;
        static BYTE parms[] =
                VTS_BSTR VTS_BSTR VTS_BSTR VTS_PVARIANT VTS_I4 VTS_I4 VTS_I4
VTS_I4 VTS_I4 VTS_I4 VTS_BSTR;
        InvokeHelper(0x5, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                lpszUsername, lpszPassword, lpszNumber, varErrorMsg, nIPAddress,
nPrimaryDNS, nSecondaryDNS, nPrimaryWINS, nSecondaryWINS, nPapChap, lpszVPN);
        return result;
}
```

DebugPrint method

Description: Used to write out to the log file.

Syntax: *object**.DebugPrint***(*nModule As Long, nLevel As Long, lpszDebug As String*)

The DebugPrint method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nModule	Long
nLevel	Long
lpszDebug	String

Sample Code using DebugPrint Method:

```
void IPhoenixWrapper::DebugPrint(long nModule, long nLevel, LPCTSTR lpszDebug)
{
        static BYTE parms[] =
                VTS_I4 VTS_I4 VTS_BSTR;
        InvokeHelper(0x1e, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 nModule, nLevel, lpszDebug);
}
```

DeleteMessage method

Description: Delete a message.

Return Type: ***A Long value.***
Syntax: *object**.DeleteMessage***(*nMsgBoxEnum As Long, nIndex As Long*)

The DeleteMessage method syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type Client.
nMsgBoxEnum	Long
nIndex	Long

Sample Code using DeleteMessage Method:

```
long IphoenixWrapper::DeleteMessage(long nMsgBoxEnum, long nIndex)
{
        long result;
        static BYTE parms[] =
                VTS_I4 VTS_I4;
        InvokeHelper(0x21, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nMsgBoxEnum, nIndex);
        return result;
}
```

Disconnect method

Description: To terminate call.

Return Type: ***A Long value.***
Syntax: *object*.***Disconnect***

The Disconnect method syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type Client.

Sample Code using Disconnect Method:

```
long IphoenixWrapper::Disconnect()
{
        long result;
        InvokeHelper(0x6, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}
```

GetAdapter method

Description: Get the name of the currently selected/active device.

Return Type: A ***String*** value.
Syntax: *object*.***GetAdapter***

The GetAdapter method syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type Client.

Sample Code using GetAdapter Method:

```
Cstring IphoenixWrapper::GetAdapter()
{
        Cstring result;
        InvokeHelper(0x1b, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
}
```

GetAdapterList method

Description: Get a list of currently available devices.

Return Type: ***A Long value.***
Syntax: *object**.GetAdapterList***(*varAdapterList As Variant*)

The GetAdapterList method syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type Client.
varAdapterList	Variant

Sample Code using GetAdapterList Method:

```
long IphoenixWrapper::GetAdapterList(VARIANT* varAdapterList)
{
        long result;
        static BYTE parms[] =
                VTS_PVARIANT;
        InvokeHelper(0x1d, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                varAdapterList);
        return result;
}
```

GetConnectStatus method

Description: Once connected, get RAS status info of the current connection.

Return Type: ***A Long value.***

Syntax:
*object**.GetConnectStatus***(*varState As Variant, varError As Variant, varBytesIn As Variant, varBytesOut As Variant, varDuration As Variant*)

The GetConnectStatus method syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type Client.
varState	Variant
varError	Variant
varBytesIn	Variant
varBytesOut	Variant
varDuration	Variant

Sample Code using ConnectStatus Method:

```
long IphoenixWrapper::GetConnectStatus(VARIANT* varState, VARIANT* varError, VARIANT*
varBytesIn, VARIANT* varBytesOut, VARIANT* varDuration)
{
        long result;
        static BYTE parms[] =
                VTS_PVARIANT VTS_PVARIANT VTS_PVARIANT VTS_PVARIANT
VTS_PVARIANT;
        InvokeHelper(0x8, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                varState, varError, varBytesIn, varBytesOut, varDuration);
        return result;
}
```

GetContact method

Description: Get the contact's name and details by index.

Return Type: ***A Long value.***
Syntax: *object***.GetContact**(*nIndex As Long, varContactName As Variant, varContactDetails As Variant*)

The GetContact method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nIndex	Long
varContactName	Variant
varContactDetails	Variant

Sample Code using GetContact Method:

```
long IphoenixWrapper::GetContact(long nIndex, VARIANT* varContactName, VARIANT*
varContactDetails)

{
        long result;
        static BYTE parms[] =
                VTS_I4 VTS_PVARIANT VTS_PVARIANT;
        InvokeHelper(0x28, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nIndex, varContactName, varContactDetails);
        return result;
}
```

GetContactInfo method

Description: Get phonebook's max size, contact name's max length, and contact detail's max length.

Return Type: ***A Long value.***
Syntax: *object***.GetContactInfo**(*varPhonebookMax As Variant, varContactNameMax As Variant, varContactDetailsMax As Variant*)

The GetContactInfo method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
varPhonebookMax	Variant
varContactNameMax	Variant
varContactDetailsMax	Variant

Sample Code using GetContactInfo Method:

```
long IphoenixWrapper::GetContactInfo(VARIANT* varPhonebookMax, VARIANT*
varContactNameMax, VARIANT* varContactDetailsMax)

{
        long result;
        static BYTE parms[] =
                VTS_PVARIANT VTS_PVARIANT VTS_PVARIANT;
        InvokeHelper(0x27, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                varPhonebookMax, varContactNameMax, varContactDetailsMax);
        return result;
}
```

GetDeviceId method

Description: Get the device ID (ESN/IMEI) of the device

Return Type: ***A String value.***
Syntax: *object**.GetDeviceId***

The GetDeviceId method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GetDeviceID Method:

```
Cstring IphoenixWrapper::GetDeviceId()
{
        Cstring result;
        InvokeHelper(0xe, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
}
```

GetDeviceModel method

Description: Get the model name of the device.

Return Type: ***A String value.***
Syntax: *object**.GetDeviceModel***

The GetDeviceModel method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GetDeviceModel Method:

```
Cstring IphoenixWrapper::GetDeviceModel()
{
        Cstring result;
        InvokeHelper(0xc, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
}
```

GetDeviceNetwork method

Description: Get currently attached network type.

Return Type: ***A String value.***
Syntax: *object**.GetDeviceNetwork***

The GetDeviceNetwork method syntax has these parts:

Part	Description

object	An expression evaluating to an object of type Client.

Sample Code using GetDeviceNetwork Method:

```
Cstring IphoenixWrapper::GetDeviceNetwork()

{
        Cstring result;
        InvokeHelper(0x3, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
}
```

GetDeviceState method

Description: Get device state. Refer to SDK.h for possible states.

Return Type: ***A Long value.***
Syntax: *object*.***GetDeviceState***

The GetDeviceState method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GetDeviceState Method:

```
long IphoenixWrapper::GetDeviceState()
{
        long result;
        InvokeHelper(0x11, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}
```

Possible States

```
typedef enum _PX_DEVICE_STATE

{

//          PX_STATE_MIN                = 0,

            PX_STATE_UNKNOWN            = 0,     // State of device cannot be determined

            PX_STATE_NOCARD             = 1,

            PX_STATE_INITIALIZING       = 2,

            PX_STATE_DISABLED           = 3,     // Disabled by Fn-F2 or user intervention

            PX_STATE_LOCKED             = 4,

            PX_STATE_SEARCHING          = 5,

            PX_STATE_IDLE               = 6,

            PX_STATE_CONNECTING         = 7,

            PX_STATE_AUTHENTICATING              = 8,

            PX_STATE_CONNECTED                   = 9,

            PX_STATE_NDIS                        = 10,
			PX_STATE_SHUTDOWN		= 11,
			PX_STATE_STANDBY		= 12,
//			PX_STATE_MAX
}PX_DEVICE_STATE;

// States that all public SMS functions will return
typedef enum {
			SMS_STATE_EMPTY = 60000,
			SMS_STATE_UNREAD,
			SMS_STATE_UNREAD_PRIORITY,
			SMS_STATE_READ,
			SMS_STATE_FORWARDED,
			SMS_STATE_REPLIED,
			SMS_STATE_SENDING,
			SMS_STATE_SENT,
			SMS_STATE_DELIVERED,
			SMS_STATE_FAILED_SEND
} SMSMessageState;

typedef enum {
			SMSInbox,
			SMSOutbox,
			SMSSentbox,
			SMSSIM
} SMSBoxEnum;
```

GetDeviceTechnology method

Description: Get device technology defined by NovatelModemAPI.h.

Return Type: ***A Long value.***
Syntax: *object*.***GetDeviceTechnology***

The GetDeviceTechnology method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GetDeviceTechnology Method:

```
long IphoenixWrapper::GetDeviceTechnology()
{
        long result;
        InvokeHelper(0x2, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}
```

GetFID method

Description: Get the FID of the device. (CDMA/EVDO Only)

Return Type: ***A String value.***
Syntax: *object**.GetFID***

The GetFID method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GetFID Method:

```
Cstring IphoenixWrapper::GetFID()
{
        Cstring result;
        InvokeHelper(0x10, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
}
```

GetHardwareVersion method

Description: Get the hardware version of the device

Return Type: ***A String value.***
Syntax: *object**.GetHardwareVersion***

The GetHardwareVersion method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GetHardwareVersion Method:

```
Cstring IphoenixWrapper::GetHardwareVersion()
{
        Cstring result;
        InvokeHelper(0xb, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
}
```

GetLockStatus method

Description: Determine whether the device is locked, including autolock setting.

Return Type: ***A Long value.***
Syntax: *object**.GetLockStatus**(varLockStatus As Variant, varAutoLockOn As Variant*)

The GetLockStatus method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
varLockStatus	Variant
varAutoLockOn	Variant

Sample Code using GetLockStatus Method:

```
long IphoenixWrapper::GetLockStatus(VARIANT* varLockStatus, VARIANT* varAutoLockOn)
{
        long result;
        static BYTE parms[] =
                VTS_PVARIANT VTS_PVARIANT;
        InvokeHelper(0x17, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                varLockStatus, varAutoLockOn);
        return result;
}
```

GetMessage method

Description: Retrieve message given which message box and an index.

Return Type: ***A Long value.***
Syntax: *object**.GetMessage**(nMsgBoxEnum As Long, nIndex As Long, varState As Variant, varMsg As Variant, nMsgSize As Long)*

The GetMessage method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nMsgBoxEnum	Long
nIndex	Long
varState	Variant
varMsg	Variant
nMsgSize	Long

Sample Code using GetMessage Method:

```
long IphoenixWrapper::GetMessage(long nMsgBoxEnum, long nIndex, VARIANT* varState,
VARIANT* varMsg, long nMsgSize)
{
        long result;
        static BYTE parms[] =
                VTS_I4 VTS_I4 VTS_PVARIANT VTS_PVARIANT VTS_I4;
        InvokeHelper(0x20, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nMsgBoxEnum, nIndex, varState, varMsg, nMsgSize);
        return result;
}
```

GetMessageCount method

Description: Get current count given which message box.

Return Type: ***A Long value.***
Syntax: *object**.GetMessageCount**(nMsgBoxEnum As Long)*

The GetMessageCount method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nMsgBoxEnum	Long

Sample Code using GetMessageCount Method:

```
long IphoenixWrapper::GetMessageCount(long nMsgBoxEnum)
{
        long result;
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x24, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nMsgBoxEnum);
        return result;
}
```

GetMessageStatus method

Description: Get a message status. Refer to SDK.h for possible states.

Return Type: ***A Long value.***
Syntax: *object**.GetMessageStatus**(nMsgBoxEnum As Long, nIndex As Long, varState As Variant)*

The GetMessageStatus method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nMsgBoxEnum	Long
nIndex	Long
varState	Variant

Sample Code using GetMessageStatus Method:

```
long IphoenixWrapper::GetMessageStatus(long nMsgBoxEnum, long nIndex, VARIANT*
varState)
{
        long result;
        static BYTE parms[] =
                VTS_I4 VTS_I4 VTS_PVARIANT;
        InvokeHelper(0x22, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nMsgBoxEnum, nIndex, varState);
        return result;
}
```

GetMobileNumber method

Description: Get the mobile number (MDN) of the device.

Return Type: ***A String value.***
Syntax: *object**.GetMobileNumber***

The GetMobileNumber method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GeMobileNumber Method:

```
Cstring IphoenixWrapper::GetMobileNumber()
{
        Cstring result;
        InvokeHelper(0xd, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
}
```

GetNetworkOperatorList method

Description: Get a list of operators. (UMTS/HSDPA Only)

Return Type: ***A Long value.***
Syntax: *object.**GetNetworkOperatorList***(*varOperatorList As Variant*)

The GetNetworkOperatorList method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
varOperatorList	Variant

Sample Code using GetNetworkOperatorList Method:

```
long IPhoenixWrapper::GetNetworkOperatorList(VARIANT* varOperatorList)
{
        long result;
        static BYTE parms[] =
                VTS_PVARIANT;
        InvokeHelper(0x1a, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                varOperatorList);
        return result;
}
```

GetNetworkPreference method

Description: Get network mode: RAT_MODE_AUTO (0), RAT_MODE_GSM (1), RAT_MODE_WCDMA (2) (UMTS/HSDPA Only)

Return Type: ***A Long value.***
Syntax: *object.**GetNetworkPreference***(*varMode As Variant*)

The GetNetworkPreference method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

varMode	Variant

Sample Code using GetNetworkPreference Method:

```
long IPhoenixWrapper::GetNetworkPreference(VARIANT* varMode)
{
        long result;
        static BYTE parms[] =
                VTS_PVARIANT;
        InvokeHelper(0x2d, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                varMode);
        return result;
}
```

GetNewMessageCount method

Description: Get new message count.

Return Type: ***A Long value.***
Syntax: *object**.GetNewMessageCount***

The GetNewMessageCount method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GetNewMessageCount Method:

```
long IPhoenixWrapper::GetNewMessageCount()
{
        long result;
        InvokeHelper(0x1f, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}
```

GetOSVersionInfo method

Description: Get the OS versioning info.

Return Type: ***A Long value.***
Syntax: *object**.GetOSVersionInfo**(varMajorVersion As Variant, varMinorVersion As Variant, varCSDVersion As Variant*)

The GetOSVersionInfo method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
varMajorVersion	Variant
varMinorVersion	Variant
varCSDVersion	Variant

Sample Code using GetOSVersionInfo Method:

```
long IPhoenixWrapper::GetOSVersionInfo(VARIANT* varMajorVersion, VARIANT*
varMinorVersion, VARIANT* varCSDVersion)
{
        long result;
        static BYTE parms[] =
                VTS_PVARIANT VTS_PVARIANT VTS_PVARIANT;
        InvokeHelper(0x9, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                varMajorVersion, varMinorVersion, varCSDVersion);
        return result;
}

CString IPhoenixWrapper::GetSoftwareVersion()
{
        CString result;
        InvokeHelper(0xa, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
}
```

GetPRLVersion method

Description: Get the PRL version of the device. (CDMA/EVDO Only)

Return Type: ***A String value.***
Syntax: *object***.GetPRLVersion***

The GetPRLVersion method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code:

```
CString IphoenixWrapper::GetPRLVersion()

{
        Cstring result;
        InvokeHelper(0xf, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
```

GetRasErrorString method

Description: Pass a RAS error code and get a RAS error string.

Return Type: ***A String value.***
Syntax: *object***.GetRasErrorString**(*nErrorCode As Long*)

The GetRasErrorString method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nErrorCode	Long

Sample Code using GetRasErrorString Method:

```
CString IPhoenixWrapper::GetRasErrorString(long nErrorCode)
```

```
{
        CString result;
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x1, DISPATCH_METHOD, VT_BSTR, (void*)&result, parms,
                nErrorCode);
        return result;
}
```

GetSigStr method

Description: Get Signal Strength. Values: 0 – 5

Return Type: ***A Long value.***
Syntax: *object*.***GetSigStr***

The GetSigStr method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GetSigStr Method:

```
long IPhoenixWrapper::GetSigStr()
{
        long result;
        InvokeHelper(0x4, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}
```

GetSoftwareVersion method

Description: Get the software (firmware) version of the device.

Return Type: ***A String value.***
Syntax: *object*.***GetSoftwareVersion***

The GetSoftwareVersion method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using GetSoftwareVersion Method:

```
CString IPhoenixWrapper::GetSoftwareVersion()
{
        CString result;
        InvokeHelper(0xa, DISPATCH_METHOD, VT_BSTR, (void*)&result, NULL);
        return result;
}
```

IsDormant method

Description: Determine whether the device is currently dormant.

Return Type: ***A Long value.***
Syntax: *object**.IsDormant***

The IsDormant method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using IsDormat Method:

```
long IPhoenixWrapper::IsDormant()
{
        long result;
        InvokeHelper(0x14, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}
```

IsMessageMemoryFull method

Description: Check to see if the message box memory is full.

Return Type: ***A Long value.***
Syntax: *object**.IsMessageMemoryFull***

The IsMessageMemoryFull method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using IsMessageMemoryFull Method:

```
long IPhoenixWrapper::IsMessageMemoryFull()
{
        long result;
        InvokeHelper(0x26, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}
```

IsRoaming method

Description: Determine whether the device is currently roaming.

Return Type: ***A Long value.***
Syntax: *object**.IsRoaming***

The IsRoaming method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using IsRoaming Method:

```
long IPhoenixWrapper::IsRoaming()
{
        long result;
        InvokeHelper(0x13, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}
```

SendMessage method

Description: To send a message.

Return Type: ***A Long value.***
Syntax: *object****.SendMessage****(varMsg As Variant, nMsgSize As Long, varMsgIndex As Variant*)

The SendMessage method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
varMsg	Variant
nMsgSize	Long
varMsgIndex	Variant

Sample Code using SendMessage Method:

```
long IPhoenixWrapper::SendMessage(VARIANT* varMsg, long nMsgSize, VARIANT*
varMsgIndex)
{
        long result;
        static BYTE parms[] =
                VTS_PVARIANT VTS_I4 VTS_PVARIANT;
        InvokeHelper(0x25, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                varMsg, nMsgSize, varMsgIndex);
        return result;
}
```

SetAdapter method

Description: To select a new active device.

Syntax: *object****.SetAdapter****(lpszAdapter As String*)

The SetAdapter method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
lpszAdapter	String

Sample Code using SetAdapter Method:

```
void IPhoenixWrapper::SetAdapter(LPCTSTR lpszAdapter)
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x1c, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                lpszAdapter);
}
```

SetAutoLock method

Description: To turn ON or OFF the autolock setting.

Return Type: ***A Long value.***
Syntax: *object***.SetAutoLock**(*nAutoOn As Long, lpszLockCode As String*)

The SetAutoLock method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nAutoOn	Long
lpszLockCode	String

Sample Code using SetAutoLock Method:

```
long IPhoenixWrapper::SetAutoLock(long nAutoOn, LPCTSTR lpszLockCode)
{
        long result;
        static BYTE parms[] =
                VTS_I4 VTS_BSTR;
        InvokeHelper(0x16, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nAutoOn, lpszLockCode);
        return result;
}
```

SetCallSettings method

Description: Set the call settings, including quality of service settings. (UMTS/HSDPA Only)

Return Type: ***A Long value.***
Syntax: *object***.SetCallSettings**(*nPDPType As Long, lpszAPN As String, nPDPAddress As*
Long, lpdQoS As Object)

The SetCallSettings method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nPDPType	Long
lpszAPN	String
nPDPAddress	Long
lpdQoS	Object

Sample Code using SetCallSettings Method:

```
long IPhoenixWrapper::SetCallSettings(long nPDPType, LPCTSTR lpszAPN, long nPDPAddress,
LPDISPATCH lpdQoS)
{
        long result;
        static BYTE parms[] =
                VTS_I4 VTS_BSTR VTS_I4 VTS_DISPATCH;
        InvokeHelper(0x12, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nPDPType, lpszAPN, nPDPAddress, lpdQoS);
        return result;
}
```

SetContact method

Description: Set the contact's name and details by index.

Return Type: ***A Long value.***
Syntax: *object***.SetContact**(*nIndex As Long, lpszContactName As String, lpszContactDetails As String*)

The SetContact method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nIndex	Long
lpszContactName	String
lpszContactDetails	String

Sample Code using SetContact Method:

```
long IPhoenixWrapper::SetContact(long nIndex, LPCTSTR lpszContactName, LPCTSTR
lpszContactDetails)
{
        long result;
        static BYTE parms[] =
                VTS_I4 VTS_BSTR VTS_BSTR;
        InvokeHelper(0x29, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nIndex, lpszContactName, lpszContactDetails);
        return result;
}
```

SetMessageStatus method

Description: Set the state of a message.

Return Type: ***A Long value.***

Syntax:
*object***.SetMessageStatus**(*nMsgBoxEnum As Long, nIndex As Long, nState As Long*)

The SetMessageStatus method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nMsgBoxEnum	Long
nIndex	Long
nState	Long

Sample Code using SetMessageStatus Method:

```
long IPhoenixWrapper::SetMessageStatus(long nMsgBoxEnum, long nIndex, long nState)
{
	long result;
	static BYTE parms[] =
		VTS_I4 VTS_I4 VTS_I4;
	InvokeHelper(0x23, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
		nMsgBoxEnum, nIndex, nState);
	return result;
}
```

SetNetworkOperator method

Description: To set the network operator provided by GetNetworkOperatorList. (UMTS/HSDPA Only)

Return Type: ***A Long value.***
Syntax: *object***.SetNetworkOperator**(*nMode As Long, nFormat As Long, lpszOperator As String*)

The SetNetworkOperator method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nMode	Long
nFormat	Long
lpszOperator	String

Sample Code using SetNeworkOperator Method:

```
long IPhoenixWrapper::SetNetworkOperator(long nMode, long nFormat, LPCTSTR lpszOperator)
{
	long result;
	static BYTE parms[] =
		VTS_I4 VTS_I4 VTS_BSTR;
	InvokeHelper(0x19, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
		nMode, nFormat, lpszOperator);
	return result;
}
```

SetNetworkPreference method

Description: Set network mode: RAT_MODE_AUTO (0), RAT_MODE_GSM (1), RAT_MODE_WCDMA (2) (UMTS/HSDPA Only)

Return Type: ***A Long value.***
Syntax: *object***.SetNetworkPreference**(*nMode As Long*)

The SetNetworkPreference method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

nMode	Long

Sample Code using SetNetworkPreference Method:

```
long IPhoenixWrapper::SetNetworkPreference(long nMode)
{
        long result;
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x2c, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nMode);
        return result;
}
```

SetProxy method

Description: Set proxy settings given a proxy IP address and port.

Return Type: ***A Long value.***
Syntax: *object***.SetProxy***(nProxy As Long, nPort As Long*)

The SetProxy method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nProxy	Long
nPort	Long

Sample Code using SetProxy Method:

```
long IPhoenixWrapper::SetProxy(long nProxy, long nPort)
{
        long result;
        static BYTE parms[] =
                VTS_I4 VTS_I4;
        InvokeHelper(0x7, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                nProxy, nPort);
        return result;
}
```

SetSMSC method

Description: Setting the SMSC is required for proper SMS functionality

Return Type: ***A Long value.***
Syntax: *object***.SetSMSC***(lpszSMSC As String*)

The SetSMSC method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
lpszSMSC	String

Sample Code using SetSMSC Method:

```
long IPhoenixWrapper::SetSMSC(LPCTSTR lpszSMSC)
{
        long result;
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x2a, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                lpszSMSC);
        return result;
}
```

Shutdown method

Description: A means of synchronously shutting down the device.

Return Type: ***A Long value.***
Syntax: *object****.Shutdown***

The Shutdown method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.

Sample Code using Shutdown Method:

```
long IPhoenixWrapper::Shutdown()
{
        long result;
        InvokeHelper(0x2b, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
        return result;
}
```

Unlock method

Description: To unlock the device. Refer to NovatelModemAPI.h for possible lock types.

Return Type: ***A Long value.***
Syntax: *object****.Unlock***(*nLockType As Long, lpszLockCode As String*)

The Unlock method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type Client.
nLockType	Long
lpszLockCode	String

Sample Code using Unlock Method:

```
long IPhoenixWrapper::Unlock(long nLockType, LPCTSTR lpszLockCode)
{
        long result;
        static BYTE parms[] =
                VTS_I4 VTS_BSTR;
        InvokeHelper(0x15, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                        nLockType, lpszLockCode);
            return result;
}
```

IEventPhoenixNotifySink object

The following section lists the event interface for Phoenix Clients. This is used for receiving server events for two-way communication. The IEventPhoenixNotifiy Sink object uses the following methods:

FireEventDeviceState method

Description: Event fired when the device state changes. Refer to SDK.h for possible states.

Syntax: *object.**FireEventDeviceState***(*nState As Long*)

The FireEventDeviceState method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type IeventPhoenixNotifySink.
nState	Long

FireEventDormant method

Description: Event fired when service is dormant. Values: 0 or 1

Syntax: *object.**FireEventDormant***(*nStatus As Long*)

The FireEventDormant method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type IEventPhoenixNotifySink.
nStatus	Long

FireEventIncomingCall method

Description: Event fired when receiving an incoming call.

Syntax: *object.**FireEventIncomingCall***(*nStatus As Long*)

The FireEventIncomingCall method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type IEventPhoenixNotifySink.
nStatus	Long

FireEventNetwork method

Description: Event fired when network service changes.

Syntax: *object.**FireEventNetwork***(*nStatus As Long*)

The FireEventNetwork method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type IEventPhoenixNotifySink.
nStatus	Long

FireEventRoaming method

Description: Event fired when service is roaming. Values: 0 or 1

Syntax: *object**.FireEventRoaming**(nRoaming As Long)*

The FireEventRoaming method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type IEventPhoenixNotifySink.
nRoaming	Long

FireEventSigStr method

Description: Event fired when the signal strength changes. Values: 0 - 5

Syntax:
*object**.FireEventSigStr**(nSigStr As Long)*

The FireEventSigStr method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type IEventPhoenixNotifySink.
nSigStr	Long

FireEventSMSStatus method

Description: Event fired when new SMS messages are available. Values: 0 or 1

Syntax: *object**.FireEventSMSStatus**(nStatus As Long)*

The FireEventSMSStatus method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type IEventPhoenixNotifySink.
nStatus	Long

QoS object

NVTL QoS Class used to set Quality of Service call settings. The QoS object uses the following properties:

deliveryofSDUError property

Description: property deliveryofSDUError

Property type: A ***Long*** value.
Syntax: *object**.deliveryofSDUError*** [= *value*]

The deliveryofSDUError property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Qos***.
value	A ***Long*** value.

deliveryOrder property

Description: property deliveryOrder

Property type: A ***Long*** value.
Syntax: *object**.deliveryOrder*** [= *value*]

The deliveryOrder property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***Long*** value.

guarBitDL property

Description: property guarBitDL

Property type: A ***Long*** value.
Syntax:
*object**.guarBitDL*** [= *value*]

The guarBitDL property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***Long*** value.

guarBitUL property

Description: property guarBitUL

Property type: A ***Long*** value.
Syntax: *object**.guarBitUL*** [= *value*]

The guarBitUL property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***Long*** value.

maxBitDL property

Description: property maxBitDL

Property type: A Long value.
Syntax: *object***.maxBitDL** [= *value*]

The maxBitDL property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***Long*** value.

maxBitUL property

Description: property maxBitUL

Property type: A ***Long*** value.
Syntax: *object***.maxBitUL** [= *value*]

The maxBitUL property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***Long*** value.

maxSDUSize property

Description: property maxSDUSize

Property type: A ***Long*** value.
Syntax: *object***.maxSDUSize** [= *value*]

The maxSDUSize property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***Long*** value.

ResBitErrorRatio property

Description: property ResBitErrorRatio

Property type: A ***String*** value.
Syntax: *object***.ResBitErrorRatio** [= *value*]

The ResBitErrorRatio property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***String*** value.

SDUErrorRatio property

Description: property SDUErrorRatio

Property type: A ***String*** value.
Syntax: *object****.SDUErrorRatio*** [= *value*]

The SDUErrorRatio property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***String*** value.

trafficClass property

Description: property trafficClass

Property type: A ***Long*** value.
Syntax:
*object****.trafficClass*** [= *value*]

The trafficClass property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***Long*** value.

trafficHandling property

Description: property trafficHandling

Property type: A ***Long*** value.
Syntax: *object****.trafficHandling*** [= *value*]

The trafficHandling property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***Long*** value.

transferDelay property

Description: property transferDelay

Property type: A ***Long*** value.
Syntax: *object****.transferDelay*** [= *value*]

The transferDelay property syntax has these parts:

Part	Description
Object	An expression evaluating to an object of type ***Qos***.
Value	A ***Long*** value.

Blaze object

NVTL Blaze ActiveX Control module uses the following events and methods:

EventDeviceState event

Syntax
Sub object_***EventDeviceState***(*nVal As Long*)

The EventDeviceState event syntax has these named arguments:

Part	Description
nVal	Long

EventDormant event

Syntax
Sub object_***EventDormant***(*nVal As Long*)

The EventDormant event syntax has these named arguments:

Part	Description
nVal	Long

EventIncomingCall event

Syntax
Sub object_***EventIncomingCall***(*nVal As Long*)

The EventIncomingCall event syntax has these named arguments:

Part	Description
nVal	Long

EventNetwork event

Syntax
Sub object_***EventNetwork***(*nVal As Long*)

The EventNetwork event syntax has these named arguments:

Part	Description
nVal	Long

EventRoaming event

Syntax
Sub object_***EventRoaming***(*nVal As Long*)

The EventRoaming event syntax has these named arguments:

Part	Description
nVal	Long

EventSigStr event

Syntax
Sub object_***EventSigStr***(*nVal As Long*)

The EventSigStr event syntax has these named arguments:

Part	Description
nVal	Long

EventSMSStatus event

Syntax
Sub object_***EventSMSStatus***(*nVal As Long*)

The EventSMSStatus event syntax has these named arguments:

Part	Description
nVal	Long

Attach method

Syntax
*object****.Attach***

The Attach method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Blaze***.

Detach method

Syntax
*object****.Detach***

The Detach method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Blaze***.

Hotspots object

NVTL Hotspots ActiveX Control Module uses the following methods:

AboutBox method

Syntax
object.***AboutBox***

The AboutBox method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Hotspots***.

Init method

Initializes Hotspot dialog.
Return Type
A ***Long*** value.
Syntax
object.***Init***

The Init method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Hotspots***.

ViewHotspots method

Shows Hotspot dialog.
Return Type
A ***Long*** value.
Syntax
object.***ViewHotspots***

The ViewHotspots method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Hotspots***.

Menu object

NVTL Menu Control uses the following methods:

Init method

Initializes language and reporting support.
Return Type
A ***Long*** value.
Syntax
*object**.Init***

The Init method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Menu***.

ShowAbout method

Shows the About dialog.
Return Type
A ***Long*** value.
Syntax
*object**.ShowAbout***

The ShowAbout method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Menu***.

ShowActivation method

Shows Activation Wizard.
Return Type
A ***Long*** value.
Syntax
*object**.ShowActivation***

The ShowActivation method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Menu***.

ShowConfig method

Shows the Configuration dialog. Contents change depending on device technology.
Return Type
A ***Long*** value.

Syntax
*object***.*ShowConfig***

The ShowConfig method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Menu***.

ShowDebug method

Shows Debug info dialog.
Return Type
A ***Long*** value.
Syntax
*object***.*ShowDebug***

The ShowDebug method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Menu***.

ShowProp method

Shows the Properties dialog. Contents change depending on device technology.
Return Type
A ***Long*** value.
Syntax
*object***.*ShowProp***

The ShowProp method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Menu***.

ShowReport method

Shows Report dialog. Captures connection statistics and logs all sessions.
Return Type
A ***Long*** value.
Syntax
*object***.*ShowReport***

The ShowReport method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Menu***.

ShowUnlock method

Shows the unlock SIM dialog. Used for PIN, PUK, and network locks.
Return Type
A ***Long*** value.
Syntax
*object****.ShowUnlock***

The ShowUnlock method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Menu***.

Language object

NVTL Language object is contained in the UtilitiesLib ActiveX Control Module. The Language Control uses the following methods:

GetLanguageCount method

Returns a total count of all supported languages.
Return Type
A ***Long*** value.
Syntax
*object****.GetLanguageCount***

The GetLanguageCount method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Language***.

GetLanguageIndex method

Returns the language index defined by standards.
Return Type
A ***Long*** value.
Syntax
*object****.GetLanguageIndex***(*nIndex As Long*)

The GetLanguageIndex method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Language***.
nIndex	Long

GetString method

Given a string id, returns the string in the currently selected language.
Return Type
A ***String*** value.
Syntax
*object**.GetString***(*lStringId As Long*)

The GetString method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Language***.
lStringId	Long

GetStringTableCount method

Returns a total count of all strings per language.
Return Type
A ***Long*** value.
Syntax
*object**.GetStringTableCount***

The GetStringTableCount method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Language***.

Init method

Initializes objects and parses all language xml files.
Return Type
A ***Long*** value.
Syntax
*object**.Init***(*lpszFilePath As String*)

The Init method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Language***.
lpszFilePath	String

ProfileManager object

NVTL ProfileManager Object is contatined in the ProfileManager ActiveX Control. The ProfileManager Control uses the following methods:

AboutBox method

Syntax
*object**.AboutBox***

The AboutBox method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***ProfileManager***.

CreateProfile method

Shows Profile Wizard given a technology.
Return Type
A ***Long*** value.
Syntax
*object**.CreateProfile***(*Technology As Long*)

The CreateProfile method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***ProfileManager***.
Technology	Long

GetDefaultProfileName method

Gets the default profile name given a technology.
Return Type
A ***String*** value.
Syntax
*object**.GetDefaultProfileName***(*Technology As Long*)

The GetDefaultProfileName method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***ProfileManager***.
Technology	Long

GetProfile method

Gets a profile. Pass in object of type. Profile.
Return Type
A ***Long*** value.
Syntax
*object**.GetProfile***(*ProfileName As String, Technology As Long, Profile As Object*)

The GetProfile method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***ProfileManager***.
ProfileName	String
Technology	Long
Profile	Object

GetProfileNameList method

Retrieves a list of profile names given a technology.
Return Type
A ***Long*** value.
Syntax
*object**.GetProfileNameList***(*varNameList As Variant, Technology As Long*)

The GetProfileNameList method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***ProfileManager***.
varNameList	Variant
Technology	Long

Init method

Initializes language support and profile database.
Return Type
A ***Long*** value.
Syntax
*object**.Init***(*ProfilePath As String*)

The Init method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***ProfileManager***.
ProfilePath	String

SetDefaultProfile method

Sets the default profile for a given technology.
Return Type
A ***Long*** value.
Syntax
*object**.SetDefaultProfile***(*ProfileName As String, Technology As Long*)

The SetDefaultProfile method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***ProfileManager***.
ProfileName	String
Technology	Long

ShowProfileList method

Shows the Profile list dialog which includes Mobile, WiFi, and Ethernet.
Return Type
A ***Long*** value.
Syntax
*object**.ShowProfileList**(Technology As Long*)

The ShowProfileList method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***ProfileManager***.
Technology	Long

Profile object

NVTL Profile Object is contatined in the ProfileManager ActiveX Control. The Profile Class has the following properties:

APN property

property APN
Property type
A ***String*** value.
Syntax
*object**.APN*** [= *value*]

The APN property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

AuthenticationType property

property AuthenticationType

Property type
A ***Long*** value.
Syntax
object.***AuthenticationType*** [= *value*]

The AuthenticationType property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

CarrierName property

property CarrierName
Property type
A ***String*** value.
Syntax
object.***CarrierName*** [= *value*]

The CarrierName property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

ConnectType property

property ConnectType
Property type
A ***Long*** value.
Syntax
object.***ConnectType*** [= *value*]

The ConnectType property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

DataSpeed property

property DataSpeed
Property type
A ***Long*** value.
Syntax
object.***DataSpeed*** [= *value*]

The DataSpeed property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

DefaultGateway property

property DefaultGateway
Property type
A ***Long*** value.
Syntax
*object**.DefaultGateway*** [= *value*]

The DefaultGateway property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

DeliveryofSDUError property

property DeliveryofSDUError
Property type
A ***Long*** value.
Syntax
*object**.DeliveryofSDUError*** [= *value*]

The DeliveryofSDUError property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

DeliveryOrder property

property DeliveryOrder
Property type
A ***Long*** value.
Syntax
*object**.DeliveryOrder*** [= *value*]

The DeliveryOrder property syntax has these parts:

Part	Description

object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

DialString property

property DialString
Property type
A ***String*** value.
Syntax
object.***DialString*** [= *value*]

The DialString property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

Fallback2GProfile property

property Fallback2GProfile
Property type
A ***String*** value.
Syntax
object.***Fallback2GProfile*** [= *value*]

The Fallback2GProfile property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

IPAddress property

property IPAddress
Property type
A ***Long*** value.
Syntax
object.***IPAddress*** [= *value*]

The IPAddress property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.

value	A ***Long*** value.

MaxSDUSize property

property MaxSDUSize
Property type
A ***Long*** value.
Syntax
*object**.MaxSDUSize*** [= *value*]

The MaxSDUSize property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

Password property

property Password
Property type
A ***String*** value.
Syntax
*object**.Password*** [= *value*]

The Password property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

PDPAddress property

property PDPAddress
Property type
A ***Long*** value.
Syntax
*object**.PDPAddress*** [= *value*]

The PDPAddress property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

PDPType property

property PDPType
Property type
A ***Long*** value.
Syntax
object.***PDPType*** [= *value*]

The PDPType property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

PrimaryDNS property

property PrimaryDNS
Property type
A ***Long*** value.
Syntax
object.***PrimaryDNS*** [= *value*]

The PrimaryDNS property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

PrimaryWINS property

property PrimaryWINS
Property type
A ***Long*** value.
Syntax
object.***PrimaryWINS*** [= *value*]

The PrimaryWINS property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

ProfileName property

property ProfileName
Property type

A ***String*** value.
Syntax
*object***.ProfileName*** [= *value*]

The ProfileName property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

ProxyAddress property

property ProxyAddress
Property type
A ***Long*** value.
Syntax
*object***.ProxyAddress*** [= *value*]

The ProxyAddress property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

ProxyPort property

property ProxyPort
Property type
A ***Long*** value.
Syntax
*object***.ProxyPort*** [= *value*]

The ProxyPort property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

ResErrorRatio property

property ResErrorRatio
Property type
A ***String*** value.
Syntax
*object***.ResErrorRatio*** [= *value*]

The ResErrorRatio property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

SDUErrorRatio property

property SDUErrorRatio
Property type
A ***String*** value.
Syntax
object.***SDUErrorRatio*** [= *value*]

The SDUErrorRatio property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

SecondaryDNS property

property SecondaryDNS
Property type
A ***Long*** value.
Syntax
object.***SecondaryDNS*** [= *value*]

The SecondaryDNS property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

SecondaryWINS property

property SecondaryWINS
Property type
A ***Long*** value.
Syntax
object.***SecondaryWINS*** [= *value*]

The SecondaryWINS property syntax has these parts:

Part	Description

object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

SetProxy property

property SetProxy
Property type
A ***Long*** value.
Syntax
*object**.SetProxy*** [= *value*]

The SetProxy property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

SMSC property

property SMSC
Property type
A ***String*** value.
Syntax
*object**.SMSC*** [= *value*]

The SMSC property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

SMSEmailNumber property

property SMSEmailNumber
Property type
A ***String*** value.
Syntax
*object**.SMSEmailNumber*** [= *value*]

The SMSEmailNumber property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

StaticIP property

property StaticIP
Property type
A ***Long*** value.
Syntax
*object**.StaticIP*** [= *value*]

The StaticIP property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

SubnetMask property

property SubnetMask
Property type
A ***Long*** value.
Syntax
*object**.SubnetMask*** [= *value*]

The SubnetMask property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

Technology property

property Technology
Property type
A ***Long*** value.
Syntax
*object**.Technology*** [= *value*]

The Technology property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

TrafficClass property

property TrafficClass
Property type
A ***Long*** value.
Syntax
object.***TrafficClass*** [= *value*]

The TrafficClass property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

TrafficHandling property

property TrafficHandling
Property type
A ***Long*** value.
Syntax
object.***TrafficHandling*** [= *value*]

The TrafficHandling property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

TransferDelay property

property TransferDelay
Property type
A ***Long*** value.
Syntax
object.***TransferDelay*** [= *value*]

The TransferDelay property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

UseDNS property

property UseDNS
Property type

A ***Long*** value.
Syntax
*object**.UseDNS*** [= *value*]

The UseDNS property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

Username property

property Username
Property type
A ***String*** value.
Syntax
*object**.Username*** [= *value*]

The Username property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

UseVPN property

property UseVPN
Property type
A ***Long*** value.
Syntax
*object**.UseVPN*** [= *value*]

The UseVPN property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

UseWINS property

property UseWINS
Property type
A ***Long*** value.
Syntax
*object**.UseWINS*** [= *value*]

The UseWINS property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***Long*** value.

VPNEntryName property

property VPNEntryName
Property type
A ***String*** value.
Syntax
*object**.VPNEntryName*** [= *value*]

The VPNEntryName property syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***Profile***.
value	A ***String*** value.

NetMonkey Lib objects

NVTL NetMonkey ActiveX Control Module Objects use the following events:

EventAdapterFound event

Event fired when interface detection state changes.
Syntax
Sub object_***EventAdapterFound***

EventAdapterUpdate event

Event fired when interface updated values are available.
Syntax
Sub object_***EventAdapterUpdate***

LAN object

NVTL LAN Object is contained in the NetMonkey ActiveX Control Module. The LAN Control uses the following methods:

GetAdapter method

Gets the currently selected interface from registry.
Return Type

A ***String*** value.
Syntax
*object**.GetAdapter***

The GetAdapter method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetAdapterList method

Gets a list of LAN interfaces available.
Return Type
A ***Long*** value.
Syntax
*object**.GetAdapterList***(*varAdapterList As Variant*)

The GetAdapterList method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.
varAdapterList	Variant

GetBytesIn method

Gets the number of bytes received.
Return Type
A ***Long*** value.
Syntax
*object**.GetBytesIn***

The GetBytesIn method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetBytesOut method

Gets the number of bytes sent.
Return Type
A ***Long*** value.
Syntax
*object**.GetBytesOut***

The GetBytesOut method syntax has these parts:

Part	Description

object	An expression evaluating to an object of type ***LAN***.

GetConnectState method

Gets the interface connection status.
Return Type
A ***Long*** value.
Syntax
object.***GetConnectState***

The GetConnectState method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetDefaultGateway method

Gets the current default gateway.
Return Type
A ***String*** value.
Syntax
object.***GetDefaultGateway***

The GetDefaultGateway method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetDuration method

Gets the duration time of the connection.
Return Type
A ***String*** value.
Syntax
object.***GetDuration***

The GetDuration method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetFriendlyName method

Gets the interface friendly name.
Return Type
A ***String*** value.

Syntax
*object***.*GetFriendlyName***

The GetFriendlyName method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetIPAddress method

Gets the current IP address.
Return Type
A ***String*** value.
Syntax
*object***.*GetIPAddress***

The GetIPAddress method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetLinkSpeed method

Gets the interface link speed in bps.
Return Type
A ***String*** value.
Syntax
*object***.*GetLinkSpeed***

The GetLinkSpeed method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetMacAddress method

Gets the interface MAC address in hex.
Return Type
A ***String*** value.
Syntax
*object***.*GetMacAddress***

The GetMacAddress method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetNdisName method

Gets the interface GUID.
Return Type
A ***String*** value.
Syntax
*object**.GetNdisName***

The GetNdisName method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

GetSubnetMask method

Gets the current subnet mask.
Return Type
A ***String*** value.
Syntax
*object**.GetSubnetMask***

The GetSubnetMask method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

Init method

Instantiates object and starts worker thread.
Return Type
A ***Long*** value.
Syntax
*object**.Init***

The Init method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***LAN***.

SetAdapter method

Sets the current interface and saves it to registry.
Syntax
*object**.SetAdapter***(*lpszAdapterNdisName As String*)

The SetAdapter method syntax has these parts:

Part	Description

object	An expression evaluating to an object of type ***LAN***.
lpszAdapterNdisName	String

WLAN object

NVTL WLAN Object is contained in the NetMonkey ActiveX Control Module. The WLAN Control uses the following methods:

AddWepKey method

Add a WEP key to an index.
Return Type
A ***Long*** value.
Syntax
*object****.AddWepKey****(nKeyIndex As Long, lpszKeyMaterial As String*)

The AddWepKey method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.
nKeyIndex	Long
lpszKeyMaterial	String

Disassociate method

Disassociates with the current access point.
Syntax
*object****.Disassociate***

The Disassociate method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

FindProfile method

Finds if a profile exists in WZConfig
Return Type
A ***Long*** value.
Syntax
*object****.FindProfile****(lpszProfile As String, nInfrastructureMode As Long*)

The FindProfile method syntax has these parts:

Part	Description

object	An expression evaluating to an object of type ***WLAN***.
lpszProfile	String
nInfrastructureMode	Long

GetAccessPoints method

Gets a list of all available access points.
Return Type
A ***Long*** value.
Syntax
*object**.GetAccessPoints***(*varAccessPoints As Variant*)

The GetAccessPoints method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.
varAccessPoints	Variant

GetAdapter method

Gets the currently selected interface from registry.
Return Type
A ***String*** value.
Syntax
*object**.GetAdapter***

The GetAdapter method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetAdapterList method

Gets a list of WLAN interfaces available.
Return Type
A ***Long*** value.
Syntax
*object**.GetAdapterList***(*varAdapterList As Variant*)

The GetAdapterList method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.
varAdapterList	Variant

GetBssid method

Gets the associated access point's MAC address.
Return Type
A ***String*** value.
Syntax
object.***GetBssid***

The GetBssid method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetBytesIn method

Gets the number of bytes received.
Return Type
A ***Long*** value.
Syntax
object.***GetBytesIn***

The GetBytesIn method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetBytesOut method

Gets the number of bytes sent.
Return Type
A ***Long*** value.
Syntax
object.***GetBytesOut***

The GetBytesOut method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetConnectState method

Gets the interface connection status.
Return Type
A ***Long*** value.
Syntax
object.***GetConnectState***

The GetConnectState method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetDefaultGateway method

Gets the current default gateway.
Return Type
A ***String*** value.
Syntax
*object**.GetDefaultGateway***

The GetDefaultGateway method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetDuration method

Gets the duration time of the connection.
Return Type
A ***String*** value.
Syntax
*object**.GetDuration***

The GetDuration method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetFriendlyName method

Gets the interface friendly name.
Return Type
A ***String*** value.
Syntax
*object**.GetFriendlyName***

The GetFriendlyName method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetIPAddress method

Gets the current IP address.
Return Type
A ***String*** value.
Syntax
object.***GetIPAddress***

The GetIPAddress method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetLinkSpeed method

Gets the interface link speed in bps.
Return Type
A ***String*** value.
Syntax
object.***GetLinkSpeed***

The GetLinkSpeed method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetMacAddress method

Gets the interface MAC address in hex.
Return Type
A ***String*** value.
Syntax
object.***GetMacAddress***

The GetMacAddress method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetNdisName method

Gets the interface GUID.
Return Type
A ***String*** value.
Syntax
object.***GetNdisName***

The GetNdisName method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetRssi method

Gets the interface signal strength in dDm.
Return Type
A ***Long*** value.
Syntax
*object**.GetRssi***

The GetRssi method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetSigStr method

Gets the interface signal strength of values 0-5.
Return Type
A ***Long*** value.
Syntax
*object**.GetSigStr***

The GetSigStr method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetSubnetMask method

Gets the current subnet mask.
Return Type
A ***String*** value.
Syntax
*object**.GetSubnetMask***

The GetSubnetMask method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetSupportedRates method

Gets interface supported rates.

Return Type
A ***String*** value.
Syntax
*object***.GetSupportedRates**

The GetSupportedRates method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

GetWZCServiceState method

Return Type
A ***Long*** value.
Syntax
*object***.GetWZCServiceState**

The GetWZCServiceState method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

Init method

Instantiates object and starts worker thread.
Return Type
A ***Long*** value.
Syntax
*object***.Init**
The Init method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

IsAdminUser method

Return Type
A ***Long*** value.
Syntax
*object***.IsAdminUser**

The IsAdminUser method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

RemoveWepKey method

Remove a WEP key from an index.
Return Type
A ***Long*** value.
Syntax
*object**.RemoveWepKey***(*nKeyIndex As Long*)

The RemoveWepKey method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.
nKeyIndex	Long

Scan method

Begins a scan for all available access points.
Syntax
*object**.Scan***
The Scan method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

SetAdapter method

Sets the current interface and saves it to registry.
Syntax
*object**.SetAdapter***(*lpszAdapterNdisName As String*)

The SetAdapter method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.
lpszAdapterNdisName	String

StartWZCService method

Syntax
*object**.StartWZCService***

The StartWZCService method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

StopWZCService method

Syntax
*object***.StopWZCService**
The StopWZCService method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WLAN***.

WWAN object

NVTL WWAN Object is contained in the NetMonkey ActiveX Control Module. The WWAN Control uses the following methods:

DisableDevice method

Disable device and turns off NDIS.
Return Type
A ***Long*** value.
Syntax
*object***.DisableDevice**(*lpszDeviceID As String*)

The DisableDevice method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.
lpszDeviceID	String

EnableDevice method

Enables devices and turns on NDIS.
Return Type
A ***Long*** value.
Syntax
*object***.EnableDevice**(*lpszDeviceID As String*)

The EnableDevice method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.
lpszDeviceID	String

GetAdapter method

Gets the currently selected interface from registry.

Return Type
A ***String*** value.
Syntax
*object****.GetAdapter***

The GetAdapter method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

GetAdapterList method

Gets a list of WWAN interfaces available.
Return Type
A ***Long*** value.
Syntax
*object****.GetAdapterList***(*varAdapterList As Variant*)

The GetAdapterList method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.
varAdapterList	Variant

GetBytesIn method

Gets the number of bytes received.
Return Type
A ***Long*** value.
Syntax
*object****.GetBytesIn***

The GetBytesIn method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

GetBytesOut method

Gets the number of bytes sent.
Return Type
A ***Long*** value.
Syntax
*object****.GetBytesOut***

The GetBytesOut method syntax has these parts:

Part	Description

object	An expression evaluating to an object of type ***WWAN***.

GetConnectState method

Gets the interface connection status.
Return Type
A ***Long*** value.
Syntax
object.***GetConnectState***

The GetConnectState method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

GetDefaultGateway method

Gets the current default gateway.
Return Type
A ***String*** value.
Syntax
object.***GetDefaultGateway***

The GetDefaultGateway method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

GetDuration method

Gets the duration time of the connection.
Return Type
A ***String*** value.
Syntax
object.***GetDuration***

The GetDuration method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

GetFriendlyName method

Gets the interface friendly name.
Return Type
A ***String*** value.

Syntax
*object***.*GetFriendlyName***

The GetFriendlyName method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

GetIPAddress method

Gets the current IP address.
Return Type
A ***String*** value.
Syntax
*object***.*GetIPAddress***
The GetIPAddress method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

GetLinkSpeed method

Gets the interface link speed in bps.
Return Type
A ***String*** value.
Syntax
*object***.*GetLinkSpeed***
The GetLinkSpeed method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

GetNdisName method

Gets the interface GUID.
Return Type
A ***String*** value.
Syntax
*object***.*GetNdisName***

The GetNdisName method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

GetSubnetMask method

Gets the current subnet mask.
Return Type
A ***String*** value.
Syntax
*object**.GetSubnetMask***

The GetSubnetMask method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

Init method

Instantiates object and starts worker thread.
Return Type
A ***Long*** value.
Syntax
*object**.Init***

The Init method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.

InitDevice method

Initializes NDIS.
Return Type
A ***Long*** value.
Syntax
*object**.InitDevice**(lpszDeviceID As String, lpszNetConnName As String, nShowIcon As Long, nReEnable As Long)*

The InitDevice method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.
lpszDeviceID	String
lpszNetConnName	String
nShowIcon	Long
nReEnable	Long

IsDeviceEnabled method

Checks the status of the NDIS device.
Return Type
A ***Long*** value.
Syntax
*object**.IsDeviceEnabled**(lpszDeviceID As String)*

The IsDeviceEnabled method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.
lpszDeviceID	String

SetAdapter method

Sets the current interface and saves it to registry.
Syntax
*object**.SetAdapter**(lpszAdapterNdisName As String)*

The SetAdapter method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.
lpszAdapterNdisName	String

UpdateDeviceParam method

Update NDIS related registry keys. eg. DialString, InitString.
Return Type
A ***Long*** value.
Syntax
*object**.UpdateDeviceParam**(lpszDeviceID As String, lpszValName As String, dwType As Long, lpszInBuffer As String, dwBuffSize As Long, dwVal As Long)*

The UpdateDeviceParam method syntax has these parts:

Part	Description
object	An expression evaluating to an object of type ***WWAN***.
lpszDeviceID	String
lpszValName	String
dwType	Long
lpszInBuffer	String

dwBuffSize	Long
dwVal	Long

Appendix H - AT Commands

Introduction

The purpose of this section is to provide advance design and integration information to assist in the integration planning and evaluation of Novatel Wireless PCI Express Mini-cards. This section is intended to specify supported AT Commands for the Novatel Wireless line of PCI Express Mini-cards, and to provide the information necessary to integrate the module into an overall product design. The full AT command list is shown for completeness. A subset of the commands shown will not be relevant to a target Mini-card.

AT Commands allow communications software to control and direct the modem. The term AT is derived from the word ***AT***tention, meaning to get the modem's attention. AT Commands are issued by an intelligent device to a modem to perform a specific function. For example, AT Commands can be used to initiate a call, answer a call or to simply transmit data. The prefix AT obtains synchronization, identifies the character parameters, and indicates that a command may be in the following characters. AT Commands ***are not*** case sensitive; use upper or lower case letters in the command syntax

The ETSI specification created a complete set of commands to interface with the terminal adapter or modem as well as specifying certain commands from ITU 25. To make the AT Command interface easier to understand and provide faster customer integration, a number of the standard AT Commands have been implemented to substitute some of the more specialized ETSI commands. All commands relevant to the control and operation of the modem have been implemented.

Novatel Wireless AT Command Set

Each of the following sections discusses an individual AT command and its five subsections listed below:

Description: describes the command and defines any conditions required to use the command

Syntax: lists the syntax the command requires

Parameters: lists any parameters and values available to the command

Response: lists and defines any responses the command may return

Example: (optional) provides examples of the command

A/

Description: REPEAT LAST COMMAND

This command repeats the last command sent to the modem.

Syntax: A/

Parameters: None

Response: Returns the command response from the previous command.

Example: ***AT+GMI***

+GMI: Novatel Wireless, Inc.

OK

A/

+GMI: Novatel Wireless, Inc.

OK

ATA

Description: ANSWER

This command sets the modem to answer the next call. The modem sends an off-hook signal to the remote station. Any additional commands on the same command line are ignored. The modem switches to data mode and after call is released, returns to command mode.

This command can be stopped by receiving a character during execution, except during some states of connection establishment such as handshaking.

Syntax: ***ATA***

Parameters: None

Response: If the connection is successful the modem will return the string:

CONNECT<text>

then switches to data mode. The <text> is only displayed if the command ATX has been set to a value greater than 0.

When the modem releases the call and returns to command mode, it returns the string:

OK

If no connection is made, the modem will return the string:

NO CARRIER

Example: ***ATA***

ATD

Description: DIAL NUMBER

This command will instruct the modem to originate a call to a disabled number. The modem attempts to set up an outgoing call.

This command can be stopped by receiving a character during execution, except during some states of connection establishment such as handshaking.

Syntax: ***ATD[<value>]***

Parameters:

<value>

The string of dialing digits and the optional V.25ter modifiers. The valid dialing digits are:

`0-9, * , #, +, A, B, C`

and the V.25ter modifiers which are ignored:

Response: If there is no dial tone, the modem will return the string:

NO DIALTONE

where the command ATX has been set to 2 or 4.

If there is an error and it is related to the mobile equipment's (ME) functionality, the modem will return the string:

+CME ERROR: <error>

If the connection is busy, the modem will return the string:

BUSY

where the command ATX has been set to 3 or 4.

If the connection cannot be established, the modem will return the string:

NO CARRIER

If the connection is successful and is non-voice call, the modem switches to data state and the modem will return the string:

CONNECT<text>

where <text> is only displayed if the command ATX has been set to a value greater than 0.

If connection successful and is a voice call, the modem will return the string:

OK

When the modem releases the call and returns to command mode, it returns the string:

OK

Example:

ATD9,555-1212	DIAL 9, PAUSE, DIAL 5551212
ATD555-1234;	DIAL 5551234
ATD#777	Packet data call

ATE

Description: ECHO MODE

This command sets the echo mode of the modem, that is, whether or not the modem echoes characters received from terminal while the modem is in command mode.

Syntax: ***ATE[<value>]***

Parameters: <value>

0	turns echo mode off

1 turns echo mode on

Response: This command will return the string OK.

Example:

ATE1	All further data entered is echoed
ATE0	All further data entered is not echoed

ATH

Description: HANG-UP (DISCONNECT)

This command instructs the modem to disconnect from the current connection and terminate the call.

Syntax: ***ATH[0]***

Parameters: 0 – This parameter is the default, as well as the only parameter, and does not necessarily have to be typed with the command.

Response: This command will return the string OK, after the Data Carrier Detect (DCD) is turned off, if it was previously on.

Example: ***ATH***

ATV

Description: VERBOSE

This command displays the modem's result codes in terse or verbose form.

In terse form, all result codes are represented by an error code number and all text messages have only a carriage return (<cr>) character appended to them.

In verbose form, all result codes are returned as words or sentences and all text messages have a carriage return-line feed (<cr><lf>) character pair before and after the text message.

Syntax: ***ATV[<value>]***

Parameters: <value>

0 will set the response format to terse

1will set the response format to verbose

Response: If the command has been set to ***0***, it will return the response:

0

If the command has been set to ***1***, it will return the response:

OK

Example: ***ATV1***

ATZ

Description: RESET PARAMETERS TO SAVED SETTINGS

This command uses the user-defined modem settings profile stored in the non-volatile memory as the active profile. If the user-defined profile is invalid, the modem will use the factory default settings. Any additional commands on the same command line are ignored. This command uses the profile created by the AT&W command.

Syntax: ***ATZ[0]***

Parameters: 0 – This parameter is the default, as well as the only parameter, and does not necessarily have to be typed with the command.

Response: This command will return the string ***OK***.

AT&C

Description: DATA CARRIER DETECT (DCD) MODE

This command sets the data carrier detect mode.

Syntax: ***AT&C[<value>]***

Parameters: <value>

0 – sets the data carrier detect so it is always on

1 – sets the data carrier detect so it is only on in the presence of a data carrier

2 – sets data carrier detect to always on but wink when connection disconnect (Qualcomm™ implemented)

Response: This command will return the string ***OK***.

Example: ***AT&C1***

AT&D

Description: DATA TERMINAL READY (DTR) CONTROL

This command defines how the modem responds, while in data mode, to the data terminal ready (DTR) circuit changing state from ON to OFF.

Syntax: ***AT&D[<value>]***

Parameters: <value>

0 – instructs the modem to ignore the data terminal ready state

1 – instructs the modem to change to command mode, while remaining connected to the call

2 – instructs the modem to disconnect from the call and then change to command mode. While the data terminal ready state is set to ***off***, the modem's auto-answer function is also off.

Response: This command will return the string ***OK***.

Example: ***AT&D2***

AT&V

Description: VIEW ACTIVE PROFILE

This command will display the active profile settings on the terminating equipment.

Syntax: ***AT&V[0]***

Parameters: 0 – This parameter is the default, as well as the only parameter, and does not necessarily have to be typed with the command.

Response: The response will be a listing of the current configuration followed by the string **OK**. For example

```
ACTIVE PROFILE :
E1 L1 M1 Q0 V1 X4 &C1 &D2
S0:0 S2:43 S3:13 S4:10 S5:8 S6:2 S7:60 S8:2 S10:15 S12:
+CBST: 7,0,1
+CSMS: 0
+CRLP: 61,61,48,6,0,3
+CRC: 0
+CR: 0
+FCLASS: 0
+IFC: 2,2
+IMODE: 0
+ICF: 3,3
+DR: 0
+CMGF: 0
+CSDH: 0
+CNMI: 2,1,0,0,0
+ILRR: 0
+IPR: 115200
+DS: 3,0,512,20
+CMEE: 0
+CREG: 0
+CCUG: 0,0,0
+CLIP: 0
+COLP: 0
+CCWA: 0
+CAOC: 1
+CLIR: 0
+CSCA: "+44385016005",145
+CSMP: 17,167

OK
```

Example: ***AT&V***

AT+CFC

Description: U_m INTERFACE FAX COMPRESSION

Syntax: ***AT+CFC=<value>***

Parameters: <value>

0 – No compression.

1 – V.42bis compression with parameters as set by the +CDS command

2 – Modified the Modified Read Compression.

Response: This command will return the string ***OK***.

Example:	***Input***	***Response***
Query	AT+CFC?	current values
Set	AT+CFC=0	OK
Test	AT+CFC=?	list of supported values

AT+CRM

Description: SET R_m INTERFACE PROTOCOL

The default value for the +CRM parameter shall be 0 if this value is supported by the MT2. If0 is not supported, the default +CRM value shall be manufacturer-specific.

Syntax: ***AT+CRM=<value>***

Parameters: <value>

0 – Asynchronous Data or Fax

1 – Packet data service, Relay Layer R_m interface

2 – Packet data service, Network Layer R_m interface, PPP.

3 – Packet data service, Network Layer R_m interface, SLIP.

4 – STU-III Service.

5-127 – Reserved for future use.

128-255 – Reserved for manufacturer-specific use.

Response: This command will return the string ***OK***.

Example:	***Input***	***Response***
Query	AT+CRM?	current values
Test	AT+CRM=?	list of supported values
Set	AT+CRM=0	OK

AT+CSQ

Description: SIGNAL QUALITY REPORT

Execution command returns received signal strength indication and channel frame error rate from the modem. Test command returns values supported by the modem.

Syntax: AT+CSQ?

Parameters: None

Response: ***+CSQ: <rssi>,<fer> OK***

RSSI (in dBm) = (<rssi> X 2) –113 (FOR 0<= <rssi> <=31)

RSSI (in dBm) = -51 (FOR 31<= <rssi> <=98)

(FOR 99<= <rssi>) rssi is not known or not detectable

FER:

0	<0.01%
1	0.01% to less than 0.1%
2	0.1% to less than 0.5%
3	0.5% to less than 1.0%
4	1.0% to less than 2.0%
5	2.0% to less than 4.0%
6	4.0% to less than 8.0%
7	>= 8.0%

99 FER not known or is not detectable

Example:	***Input***	***Response***
Test	AT+CSQ?	list of RSSI and FER OK

AT+CSS

Description: REPORT SERVING SYSTEM INFORMATION

Syntax: ***AT+CSS?***

Parameters: None

Response: ***+CSS: <Band Class>,<Band>,<SID> OK***

Band Class:

0	The mobile station is registered with a cellular system.
1	The mobile station is registered with a PCS system.

Band:

0	The mobile station is registered with a PCS A-band system.
1	The mobile station is registered with a PCS B-band system.
2	The mobile station is registered with a PCS C-band system.
3	The mobile station is registered with a PCS D-band system.
4	The mobile station is registered with a PCS E-band system.
5	The mobile station is registered with a PCS F-band system.
6	The mobile station is registered with a cellular A-band system.
7	The mobile station is registered with a cellular B-band system.
8	The mobile station is not registered.

SID:0-16383	The mobile station is registered with the system indicated.
99999	The mobile station is not registered.

Example:	***Input***	***Response***
Test	AT+CSS?	+CSS: 1,0,1031 OK

AT+CXT

Description: ACTION FOR AN UNRECOGNIZED COMMAND

Syntax: ***AT+CXT=<value>***

Parameters: <value>

0 Do not pass unrecognized commands to the IWF.

1 When detecting an unrecognized AT command, open transport layer connection and pass unrecognized command to the IWF.

Example:	***Input***	***Response***
Query	***AT+CXT?***	***+CXT: 0 OK***
Set	***AT+CXT=0***	***OK***

AT+ER

Description: ERROR CONTROL REPORTING

This extended-format numeric parameter controls whether the extended-format +ER: intermediate result code is transmitted from the IWF over the U_m interface.

Syntax: ***AT+ER=<value>***

Parameters: <value> should be referred to IS-131.

Response: This command will return the string ***OK***.

Example:	***Input***	***Response***
Query	AT+ER?	current values
Set	AT+ER=0	OK

AT+ETBM

Description: CONTROLS THE HANDLING OF DATA REMAINING IN IWF BUFFERS

This extended-format compound parameter controls the handling of data remaining in IWF buffers upon service termination.

Syntax: ***AT+ETBM=<value>***

Parameters: <value> should be referred to IS-131.

Response: This command will return the string ***OK***.

Example:	***Input***	***Response***
Query	AT+ETBM?	current values
Set	AT+ETBM=1,1,20	OK

AT+FCLASS

Description: FAX CLASS SET OR TEST

Sets a particular mode of operation (data, fax). This causes the TA to process information in a manner suitable for that specific type of device.

Syntax: ***AT+FCLASS[=<value> or ?][?]***

Parameters: <value>, [?]

0	data
2.0	fax class 2 (TIA-578-A)
?	queries the command and returns its current setting or displays the valid values for the commands parameters

Response: This command will return the string ***OK***.

Example:	***Input***	***Response***
Query	AT+FCLASS?	current values
Test	AT+FCLASS=?	list of supported values
Set	AT+FCLASS=0	OK

AT+GCAP

Description: REPORT ADDITIONAL CAPABILITIES

Modem reports a list of additional capabilities.

Syntax: ***AT+GCAP***

Parameters: None

Response: This command will return the string ***OK***.

Example:	***Input***	***Response***
Query	AT+GCAP	+GCAP: +CIS707-A, +MS, +ES, +DS, +FCLASS OK

AT+GMI

Description: MANUFACTURER IDENTITY

Request for manufacturer identification

Syntax: ***AT+GMI***

Parameters: None

Response: ***+GMI: <MANUFACTURERS NAME/ID> OK***

Example:	***Input***	***Response***
Query	AT+GMI	+GMI: Novatel Wireless Inc. OK

AT+GMM

Description: MODEM IDENTITY

Request TA model identification. Unit reports one or more lines of information text which permits you to identify the specific model of device. Typically, the text will consist of a single line containing the name of the product, but manufacturers may choose to provide any information desired.

Syntax: ***AT+GMM***

Parameters: None

Response: ***+GMM: <MODEL ID> OK***

Example:	***Input***	***Response***
Query	AT+GMM	EXPDV620

AT+GMR

Description: REVISION NUMBER / IDENTITY

This command reports the version, revision and date of the software or firmware used in the device. It is also used to identify the software version to facilitate easier tracking and code updates.

Syntax: ***AT+GMR***

Parameters: None

Response: ***+GMR: <REVISION ID> OK***

Example:	***Input***	***Response***
Query	AT+GMR	M6500C-NIRVANA_VZW-Q40305.136 [Mar 22 2005 14:00:00]

AT+GSN

Description: ESN NUMBER IDENTITY

This command causes the MT2 to transmit one or more lines of information text, determined by the manufacturer, which is intended to permit you of the MT2 to identify the individual device. Typically, the text will consist of a single line containing a manufacturer-determined alphanumeric string, but manufacturers may choose to provide any information desired.

Syntax: ***AT+GSN***

Parameters: None

Response: ***+GSN: <ESN(hex)> OK***

Example:	***Input***	***Response***
Query	AT+GSN	+GSN: <ESN(hex)> OK

AT$NW

Description: This command returns Novatel Wireless company description

Syntax: ***AT$NW***

Parameters: None

Example:	***Input***	***Response***
Query	AT$NW	Novatel Wireless Inc. www.novatelwireless.com

Developed in the USA.

AT$NVTLLTIME

Description: This command returns the local time received from the network and time zone offset. If there is no service available then the time reported starts from 1980/01/06

Syntax: ***AT$NVTLLTIME***

Parameters: None

Example:	***Input***	***Response***
Query	AT$NVTLLTIME	2005.5.18.16.39.0.2.-7 OK

AT$NVTLMDN

Description: This command returns the mobile directory number of the device.

Syntax: ***AT$NVTLMDN***

Parameters: None

Example:	***Input***	***Response***
Query	AT$NVTLMDN	0000000140 OK

AT+IOTA

Description: This command is used to enable/disable/start eIOTA. (Only available on the Sprint PCS Network)

Syntax: ***AT+IOTA***

Parameters:

QUERY TEST, AT+IOTA=?

This command returns the range that the command supports (0-2).

ARGUMENT, AT+IOTA=<value>

This command setup the IOTA setting.

Value:

0	-	Disable eIOTA
1	-	Enable eIOTA
2	-	Start eIOTA session

QUERY, AT+IOTA?

This command returns the current status during IOTA session.

Status format:

IOTA Enabled or IOTA Disabled

In Progress: <x>	(0 = not in progress, 1 = in progress)
Repeat Test OK: <x>	(x = number of repeat test OK)
Repeat Test Failed: <x>	(x = number of repeat test failed)

Retry Command: <x> (x = retry command in numeric)

Current State: <x> (x = current IOTA state in numeric)

Network Up or Network Down

Server Connected or Server Disconnected

Retry: <x> (x = the number of retry IOTA session)

Global State: <x>

Number Get: <x>

Number Post: <x>

Proxy Trusted or Proxy Not Trusted

AT$NWACTIVATION

Description: This command is used to manually provision the Novatel PCMCIA card (program the MDN and MIN).

Syntax: ***AT$NWACTIVATION***

Parameters:

QUERY TEST, AT$NWACTIVATION =?

This command returns the range that the following string:

$NWACTIVATION: (ACTIVATION CODE:[xxxxxx] MDN:[XXXXXXXXXX]
MIN:[XXXXXXXXXX])

ARGUMENT, AT$NWACTIVATION = <SPC>, <MDN>, <MIN>

This command programs the MDN and MIN into the PCMCIA card using the correct SPC.

Value:

SPC	-	Service Programming Code (6 digits)
MDN	-	Mobile Directory Number (10 digits)
MIN	-	Mobile Identification Number (10 digits)

QUERY, AT$NWACTIVATION?

This command returns the MDN, Min1, and MIN2.

AT+PZID

Description: This command is used to retrieve the PZID of the current network. Values are only 1 and 0 being non zero or else zero. (Only available on the Sprint PCS Network)

Syntax: ***AT+PZID***

Parameters:

QUERY, AT+PZID?

This command returns the current PZID of serving network.

Status format:

PZID - 1 for non zero value of PZID

0 for zero value of PZID

AT$SPNAI

Description: This command is used to determine if the current device supports 6 MIP profiles. (Only available on the Sprint PCS Network)

Syntax: ***AT$SPNAI***

Parameters:

QUERY, AT$SPNAI?

This command returns the state if the device supports 6 MIP profiles.

Status format:

SPNAI	1 current device does support 6 MIP profiles
	0 does not support 6 MIP profiles

Appendix I - Novatel Wireless Developer Network Library

The NWDN Library is a comprehensive reference for developers writing applications for Novatel Wireless modems. It contains software API references, modem AT command set references, and any other documentation you might need to develop solutions that use Novatel Wireless modems.

AT+COPS

Description:

Set command forces an attempt to select and register the GSM network operator. <mode> is used to select whether the selection is done automatically by the ME or is forced by this command to operator <oper> (it shall be given in format <format>). If the selected operator is not available, no other operator shall be selected (except <mode>=4). The selected operator name format shall apply to further read commands (+COPS?) also. <mode>=2 forces an attempt to deregister from the network. The selected mode affects to all further network registration (e.g. after <mode>=2, ME shall be unregistered until <mode>=0 or 1 is selected). Refer subclause 9.2 for possible <err> values. This command should be abortable when registration/deregistration attempt is made.
Read command returns the current mode and the currently selected operator. If no operator is selected, <format> and <oper> are omitted.
Test command returns a list of quadruplets, each representing an operator present in the network. Quadruplet consists of an integer indicating the availability of the operator <stat>, long and short alphanumeric format of the name of the operator, and numeric format representation of the operator. Any of the formats may be unavailable and should then be an empty field. The list of operators shall be in order: home network, networks referenced in SIM, and other networks. It is recommended (although optional) that after the operator list TA returns lists of supported <mode>s and <format>s. These lists shall be delimited from the operator list by two commas.

Syntax:

Command	Response
+COPS=[<mode>[,<format>[,<oper>]]]	+CME ERROR: <err>
+COPS?	+COPS: <mode>[,<format>,<oper>[,<AcT]] +CME ERROR: <err>
+COPS=?	+COPS: [list of supported (<stat>,long alphanumeric <oper>,short alphanumeric <oper>,numeric <oper>[,<AcT>])s] [,,(list of supported <mode>s),(list of supported <format>s)] +CME ERROR: <err>

Values:

<mode>

Value	Description
0	automatic (<oper> field is ignored)
1	manual (<oper> field shall be present)
2	deregister from network
3	set only <format> (for read command +COPS?), do not attempt registration/deregistration (<oper> field is ignored); this value is not applicable in read command response
4	manual/automatic (<oper> field shall be present); if manual selection fails, automatic mode (<mode>=0) is entered ***(NOT SUPPORTED)***

<format>

Value	Description
0	long format alphanumeric <oper>
1	short format alphanumeric <oper>
2	numeric <oper> <oper>: string type; <format> indicates if the format is alphanumeric or numeric; long alphanumeric format can be up to 16 characters long and short format up to 8 characters (refer GSM MoU SE.13 [9]); numeric format is the GSM Location Area Identification number (refer GSM 04.08 [8] subclause 10.5.1.3) which consists of a three BCD digit country code coded as in ITU-T E.212 Annex A [10], plus a two BCD digit network code, which is administration specific; returned <oper> shall not be in BCD format, but in IRA characters converted from BCD; hence the number has structure: (country code digit 3)(country code digit 2)(country code digit 1)(network code digit 2)(network code digit 1)

<stat>

Value	Description
0	Unknown
1	Available
2	Current
3	Forbidden

<AcT>

Value	Description
0	GSM
1	GSM Compact (Not Supported)
2	UTRAN
3	Automatic

Note:

<AcT> value of 3 (automatic) is an added feature not supported by 3GPP.

Example:

AT+COPS?
AT+COPS=1,1,"T-Mobile"

AT+CSPN

Description:

Returns the current service provider name.

Syntax:

Command	Response
+CSPN?	+CSPN: <oper>
+CSPN=?	ERROR, +CME ERROR: <err>
+CSPN	ERROR, +CME ERROR: <err>

Values:

<oper>

String type; long alphanumeric format can be up to 16 characters long

Example:

AT+CSPN?

AT$NWATR

Description:

This command allows the user to read the ATR (answer-to-reset) string from the SIM. Used for AT+CSIM to determine the capabilities of the SIM. The ATR string is described in ISO/IEC 7816-3 as mentioned in ETSI 31.111.

Syntax:

Command	Response
$NWATR?	$NWATR: <length>, <atr_string>
$NWATR=?	ERROR, +CME ERROR: <err>
$NWATR=	ERROR, +CME ERROR: <err>
$NWATR	ERROR, +CME ERROR: <err>

Values:

<length>
Length of <atr_string>.

<atr_string>
string of hex characters as described in ISO/IEC 7816-3.

Example:

AT$NWATR?

AT$NWCID

Description:

Read command returns the current serving cell ID and LAC information.

Syntax:

Command	Response
$NWCID?	$NWCID: <cell id>,<lac>
$NWCID=?	ERROR
$NWCID	ERROR

Note:

$NWCID command is only available in BUILD 38 (U530) and BUILD 10 (U630) or greater.
If the UE has not camped on a cell then the read command returns “Unknown.”

Example:

AT$NWCID

AT$NWFLASH

Description:

Read command returns the memory device that is populated on the PCB.

Syntax:

Command	Response
$NWFLASH?	$NWFLASH: <id 0> <id 1> <id 2> <id 3>
$NWFLASH=?	ERROR
$NWFLASH	ERROR

Note:

$NWFLASH command is only available on U630 devices.

Example:

AT$NWFLASH?

AT$NWHLR

Description:

The HLR Lock feature, when enabled, allows service providers to limit a UE's roaming area (country and network specific) as well as prohibit unauthorized or stolen SIMs from operating within their network.

The HLR Lock feature utilizes a portion of the SIM International Mobile Subscriber Identity (IMSI) number to carry out the algorithm. The makeup of the IMSI number is composed of a 3 digit MCC value, a 2 (or 3) digit MNC value and a 9 digit MSIN value. The first three leading digits of the MSIN form the HLR-value (2 digits) and a NDC identity (1 digit).

The MCC/MNC pair, NDC identity and HLR range(s) that are programmed within the UE are secured by a service provider specified password.

Syntax:

Command	Response
$NWHLR=(0,<mcc>,<mnc>)|(1,<mode>,<parm>,[hlr rangelow>,<hlr rangehigh>,…]|(2)|(3,1,<passwd>)|(4,<type>)	$NWHLR: <mcc>,<mnc> $NWHLR: <index> <ndc> <list of hlr ranges> OK, ERROR, +CME ERROR: <err>
$NWHLR=?	ERROR
$NWHLR?	ERROR

Values:

<cmd>

Parameter specifies the desired command.

Value	Description
0	Write new MCC/MNC value
1	Write new OR append to OR delete an HLR entry
2	Erase all HLR entries
3	Set Password/Enable HLR feature
4	Read HLR entries

<mode>

Parameter specifies the desired write operation within the specified command.

Value	Description
0	Write a new HLR entry
1	Append to an existing HLR entry
2	Erase an existing HLR entry

<type>

Parameter specifies the desired read operation within the specified command.

Value	Description
0	Read MCC/MNC entry
1	Read existing HLR list(s)

<mcc>

Parameter specifies the mobile country code.

<mnc>

Parameter specifies the mobile network code.

<parm>

Parameter specifies a valid NDC digit or valid HLR list INDEX

<passwd>

Parameter specifies the 12 digit alphanumeric password required to lock/unlock the HLR lists. Once the password is set, the user must use the facility lock (***AT+CLCK***) to unlock the HLR codes.

<hlr range$_{low}$>

Parameter specifies the numeric low value of the acceptable HLR value. Acceptable entries range from 0 to 99 inclusive.

<hlr range$_{high}$>

Parameter specifies the numeric high value of the acceptable HLR value. Acceptable entries range from 0 to 99 inclusive.

Note:

A maximum of 4 NDC values with 15 corresponding HLR range specifications may be entered. Only one MCC/MNC pair is supported.

Example:

The HLR Lock codes can be read, set, erased and locked using the following custom AT command:

```
AT$NWHLR=<cmd>,[<item1>,<item2>,…]
```

Where <***cmd>*** is one of the following:
0 : for entry of MCC/MNC pair
1 : for entry/appending/deleting of NDC/HLR values
2 : for deletion of entire HLR entries (including MCC/MNC pair)
3 : for password entry and activation of HLR Lock
4 : for displaying HLR entries

Where <***item1>,<item2>,…*** represents action items corresponding to the requested ***<cmd***> (see below for further details)

If ***< cmd >*** = 0,
<mcc>,<mnc> is comma-delimitated MCC/MNC network identifier.
(only one entry supported with this command)

If ***< cmd >*** = 1,
<mode>,<parm>,[<hlr range$_{low}$>,<hlr range$_{high}$>…] is comma-delimitated for entry, appending or deleting HLR parameters.

If ***< mode >*** = 0,
<parm> = NDC digit (IMSI 8th digit if MNC is 2 digits in length or IMSI 9th digit if MNC is 3 digits in length)
<hlr range$_{low}$>,<hlr range$_{high}$> is comma-delimitated service provider specified ranges (valid entries are from 0 to 99). Up to 5 HLR range pairs (low/high) may be entered per <mode>=0 command.

If ***< mode >*** = 1,
<parm> = index (0,1,2,3 to an existing HLR list)
<hlr range$_{low}$>,<hlr range$_{high}$> is comma-delimitated service provider specified ranges (valid entries are from 0 to 99). Up to 5 HLR range pairs (low/high) may be entered for appending to an existing list (per command).

If ***< mode >*** = 2,
<parm> = index (0,1,2,3 to delete a single HLR list entry)

If ***< cmd >*** = 2 entire HLR entries deleted (Index 0-3 and MCC/MNC pair)

If ***< cmd >*** = 3,1, To enable & lock the HLR feature.
<passwd> = "xxxxxxxxxxxx" up to 12 digits in length.

If ***< cmd >*** = 4,
<type> = 0 to display the single MCC/MNC entry.
<type> = 1 to display NDC and HLR range values.

To populate the network personalization code for the HLR Lock feature (lock to network 123 02):
AT$NWHLR=0,123,02

To populate a new NDC value of 7 and the following HLR ranges; 10-15, 20-22, 34-38, and 67-70:
AT$NWHLR=1,0,7,10,15,20,22,34,38,67,70

AT$NWICCID

Description:

Read command returns the SIM ICC ID.

Syntax:

Command	Response
$NWICCID?	$NWICCID: <iccid>
$NWICCID=?	ERROR
$NWICCID	ERROR

Note:

$NWICCID command is only available on HSDPA devices.

Example:

AT$NWICCID?

AT$NWNPC

Description:

This command allows the user to read, set, and erase the network personalization codes. The network personalization codes are stored on each card. The purpose is to allow the card to function only if the MNC/MCC list on the SIM matches the list stored on the card.

Syntax:

Command	Response
$NWNPC?	ERROR, +CME ERROR: <err>
$NWNPC=?	$NWNPC: (list of supported <mode>s)
$NWNPC=0|(1,<mcc>, <mnc>)|(2, <index>)|(3, <passwd>)	$NWNPC: <mcc>,<mnc>
$NWNPC	ERROR, +CME ERROR: <err>

Values:

<mode>

Parameter specifies valid <mode>s.

Value	Description
0	Read current NPC list
1	Write new NPC entry
2	Erase NPC entry
3	Set password

<mcc>

Parameter specifies the mobile country code.

<mnc>
Parameter specifies the mobile network code.

<index>
Parameter specifies a valid NPC index number.

<passwd>
Parameter specifies the 12 digit alphanumeric password required to unlock the network personalization code. Once the password is set, the user must use the facility lock (AT+CLCK) to unlock the network personalization codes.

Note:

A maximum of 30 NPC entries are allowed to be entered. This command is only enabled in TEST state.

Example:

AT$NWNPC=0

See Also:

AT+CLCK

AT$NWPDN

Description:

This command performs an orderly shutdown of the modem saving the current MRU settings.

Syntax:

Command	Response
$NWPDN	OK

Note:

$NWPDN Command only available in Build 34 or greater.
Once the $NWPDN is issued, the OK response is returned upon completion of the command. Upon completion of the command, the card must then be power cycled before it is operational again. It is expected that $NWPDN be the last command issued by a modem manager before it removes power from the device.

Example:

AT$NWPDN

AT$NWPINR

Description:

This command allows the user to read the number of incorrect PIN entries remaining on the SIM before PUK lock is enabled.

Syntax:

Command	Response
$NWPINR?	$NWPINR: <num_of_retries>
$NWPINR=?	ERROR, +CME ERROR: <err>

$NWPINR=	ERROR, +CME ERROR: <err>
$NWPINR	ERROR, +CME ERROR: <err>

Values:

None.

Note:

This command should be used after AT+CPIN? To verify that the SIM PIN is requested. If the SIM is already unlocked and SIM PIN entry is not necessary then the command does not return a valid number of retries.

Example:

AT$NWPINR?

AT$NWRAT

Description:

Set command controls the preferred Radio Access Technology to be used by the modem.
Read command returns the preferred and current Radio Access Technology being employed by the modem.

Syntax:

Command	Response
$NWRAT?	$NWRAT: <mode>,<domain>,<state>
$NWRAT=?	$NWRAT: (list of supported <mode>s, list of supported <domain>s)
$NWRAT=<mode>,<domain>	OK, ERROR, +CME ERROR: <err>

Values:

<mode>

Value	Description
0	Automatic
1	GSM Only
2	WCDMA Only

<domain>

Value	Description
0	CS Only (Circuit Switched)
1	PS Only (Packet Swicthed)
2	CS+PS

<state>

Value	Description
0	Searching
1	WCDMA CS
2	WCDMA PS
3	WCDMA CS+PS
4	GSM CS
5	GSM PS
6	GSM CS+PS

Note:

$NWRAT Command only available in Build 20 or greater.
When switching the service domain within a specific mode (RAT) the modem would not change its service domain unless it lost coverage or changed modes. For example, if the modem is GSM PS and the following command is issued AT$NWRAT=1,0 to change to GSM CS, the change will not occur until loss of coverage or change of RAT.

Example:

AT$NWRAT=2,2

Appendix J - Additional AT Commands

This section provides the additional details of the ETSI AT command set implementation for capable mobile equipment. This implements a minimally featured data-capable WCDMA and GSM ASIC that performs the needed circuit-switched and packet-switched (PDP type PPP and GPRS) service.

A series of tables lists these commands, with the first table describing the type of information provided in each column. Unless specifically noted in the command description, all commands listed in the following tables are rejected by the command processor when the SIM is absent or when SIM PIN validation is pending.

Each of the following sections discusses an individual AT command in the subsections below:

Description:	Describes the command. Defines any conditions required to use the command
Syntax:	Lists the syntax the command requires
Parameters:	Lists any parameters and value ranges for the command
Implementation:	Explains whether QUALCOMM™ has implemented the command

ATH

Description:	Hook control command to terminate call in progress. Does not terminate voice calls.
Syntax:	ATH<value>
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Fully. Online command mode only supported for Async data

ATI

Description:	Request identification information This extended-format compound parameter is used to control t he operation of local flow control between the DTE and DCE.
Syntax:	ATI
Parameters:	No value accepted
Implementation:	Fully. Unit outputs: manufacturer, model number, mobile software revision, boot block version, release date, release time, IMEI, complete capabilities list

ATL

Description:	Monitor speaker loudness
Syntax:	ATL<value>
Parameters:	<value>

	Values per Spec ITU-T V.25ter
Implementation:	Command accepted, no action taken. Mobile audio stream not used for Async data

ATO

Description:	Return to online data state from online command state
Syntax:	ATO<value>
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Fully. Online command mode only supported for Async data

ATP

Description:	Select pulse dialing
Syntax:	ATP
Parameters:	
Implementation:	Command accepted, performs normal dial. Pulse dialing not relevant to ETSI data services. ‘P’ not sent in dial string.

ATQ

Description:	Result code suppression
Syntax:	ATQ[<value>]
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Fully

ATS0

Description:	Enable/disable automatic answering
Syntax:	ATS0=<value>
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Fully.

ATS3

Description:	Command line termination character
Syntax:	ATS3

Parameters:	Values per Spec ITU-T V.25ter
Implementation:	Fully

ATS4

Description:	Response formatting character
Syntax:	ATS4
Parameters:	Values per Spec ITU-T V.25ter
Implementation:	Fully

ATS5

Description:	Command line editing character
Syntax:	ATS5
Parameters:	Values per Spec ITU-T V.25ter
Implementation:	Fully

ATS6

Description:	Pause before blind dialing
Syntax:	ATS6=<value>
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Command accepted, no effect on data call. Not applicable to wireless call.

ATS7

Description:	Number of seconds to establish end-to-end data connection
Syntax:	ATS7=<value>
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Command accepted, no effect on data call. Async data command.

ATS8

Description:	Number of seconds to pause when “.” is encountered in dial string
Syntax:	ATS8=<value>
Parameters:	<value> Values per Spec ITU-T V.25ter

Implementation:	Command accepted, no effect on data call. Async data command.

ATS10

Description:	Number of tenths of a second from carrier loss to disconnect
Syntax:	ATS10=<value>
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Command accepted, no effect on data call. Async data command.

ATT

Description:	Select tone dialing
Syntax:	ATT
Parameters:	
Implementation:	Command accepted, performs normal dial. Tone dialing not relevant to ETSI data services. 'T' not sent in data string.

ATX

Description:	Result code selection and call progress monitoring control
Syntax:	ATX[<value>]
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Command accepted, no action taken

AT&F

Description:	Set to Factory defined configuration (effect is implementation-dependent)
Syntax:	AT&F<value>
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Fully. Same behaviour as ATZ except it changes baud rate to default value.

AT+CBC

Description:	Report battery charge
Syntax:	AT+CBC
Parameters:	

Implementation:	Fully.

AT+CBST

Description:	Selects the circuit-switched bearer service with data rate and connection element when data calls are originated
Syntax:	AT+CBST=<speed>,<name>,<ce>
Parameters:	<name> 0, data circuit asynchronous 1, data circuit synchronous 4, data circuit asynchronous (RDI) <speed> (in bps) 0, autobaud 7, 9600 (V.32) 12, 9600 (V.34) 14, 14400 (V.34) 16, 28800 (V.34) 17, 33600 (V.120) 39, 9600 (v.120) 43, 14400 (v.120) 48, 28800 (v.120) 51, 48000 (v.120) 71, 9600 (v.110) 75, 14400 (v.120) 80, 28800 (v.110) 81, 38400 (v.110) 83, 56000 (x.31 flag stuffing, UDI/RDI) 84, 64000 (x.31 flag stuffing, UDI) 116, 64000 134, 64000 (multimedia) <ce> 0, data transparent 1, data nontransparent * setting can be used in conjunction with asynchronous non-transparent UDI/RDI service in order to get Frame Tunneling mode.
Implementation:	Fully. In WCDMA mode only, data circuit synchronous UDI service is supported

AT+CCFC

Description:	Controls call forwarding supplementary service
Syntax:	AT+CCFC=<reason>, <mode>, <number>, <type>, <class>, <subaddr>, <satype>, <time>
Parameters:	Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CCUG

Description:	Controls closed user group supplementary service
Syntax:	AT+CCUG=<n>, <index>, <info>
Parameters:	Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CCWA

Description:	Control of Call Waiting Supplementary Services
Syntax:	AT+CCWA=[<n>[,<mod>[,<class>]]]
Parameters:	Command input. Unsolicited result code <class>: only 1 or 2 reported. Unsolicited result code optional <alpha> and <CLI validity> not supported Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CFUN

Description:	Sets the level of functionality in the ME
Syntax:	AT+CFUN=[<fun>[,<rst>]]
Parameters:	<fun>: 0, 1 Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CGACT

Description:	Activate or deactivate the specified PDP context(s)
Syntax:	AT+CGACT
Parameters:	<cid>: 1 to 16 Values per Spec 3GPP TS 27.007
Implementation:	Fully

AT+CGATT

Description:	Attach or detach from the Packet Domain Service
Syntax:	AT+CGATT
Parameters:	Values per Spec 3GPP TS 27.007
Implementation:	Fully

AT+CGCLASS

Description:	Set the GPRS mobile class
Syntax:	AT+CGCLASS
Parameters:	Values per Spec 3GPP TS 27.007
Implementation:	Command accepted, no actions taken

AT+CGDCONT

Description:	Set PDP context parameter values for a PDP context identified by connection identifier
Syntax:	AT+CGTDCONT=<cid>,<PDP_Type>,<APN>,<PDP_addr>,<d_comp>, <h_comp>
Parameters:	<cid>: 1 to 16 <PDP_type>: IP, PDP-IP, PPP, PDP-PPP <d_comp>: 0, 1 <h_comp>: 0, 1 Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CGDSCO

Description:	Define Secondary PDP Context
Syntax:	AT+CGDSCO=[<cid>,<p_cid>[,<d_comp>[,<h_comp>]]]
Parameters:	<cid>: 1 to 16 <p_cid>: 1 to 16 <d_comp>: 0, 1 <h_comp>: 0, 1 Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CGEQMIN

Description:	Set the minimum acceptable UMTS QoS Profile against the negotiated profile in Activate PDP Context Request message
Syntax:	AT+CGEQMIN=<cid>, <Traffic_class>, <maximum_bitrate_UL>, <maximum_bitrate_DL>, <Guranteed_bitrate_UL>, <Guaranteed_bitrate_DL>, <Delivery_order>, <Maximum_SDU_size>, <SDU_error_ratio>, <Residual_bit_error_ratio>, <Delivery_of_erroneous_SDUs>, <Transfer_delay>, <Traffic_handling_priority>

Parameters:	<cid>: 1 to 16 <Traffic_class>: 0 to 16 <maximum_bitrate_UL>: 0 to 384 <maximum_bitrate_DL>: 0 to 384 <Guranteed_bitrate_UL>: 0 to 384 <Guaranteed_bitrate_DL>: 0 to 384 <Delivery_order>: 0 to 2 <Maximum_SDU_size>: 0 to 1520 <SDU_error_ratio>: 0E0, 1E1, 1E2, 7E3, 1E3, 1E4, 1E5, 1E6 <Residual_bit_error_ratio>: 0E0, 5E2, 1E2, 5E3, 4E3, 1E3, 1E4, 1E5, 1E6, 6E8 <Delivery_of_erroneous_SDUs>: 0 to 3 <Transfer_delay>: 0, 100 to 4000 <Traffic_handling_priority>: 0 to 3 Set values are saved across power cycles Values per Spec 3GPP TS 27.007
Implementation:	Fully. Setting these parameters will reset +CGQMIN and +CGQREQ to defaults.

AT+CGEQREQ

Description:	Set the UMTS QoS Profile that is used in Activate PDP Context Request message
Syntax:	AT+CGEQREQ=<cid>, <Traffic_class>, <maximum_bitrate_UL>, <maximum_bitrate_DL>, <Guranteed_bitrate_UL>, <Guaranteed_bitrate_DL>, <Delivery_order>, <Maximum_SDU_size>, <SDU_error_ratio>, <Residual_bit_error_ratio>, <Delivery_of_erroneous_SDUs>, <Transfer_delay>, <Traffic_handling_priority>
Parameters:	<cid>: 1 to 16 <Traffic_class>: 0 to 16 <maximum_bitrate_UL>: 0 to 384 <maximum_bitrate_DL>: 0 to 384 <Guranteed_bitrate_UL>: 0 to 384 <Guaranteed_bitrate_DL>: 0 to 384 <Delivery_order>: 0 to 2 <Maximum_SDU_size>: 0 to 1520 <SDU_error_ratio>: 0E0, 1E1, 1E2, 7E3, 1E3, 1E4, 1E5, 1E6 <Residual_bit_error_ratio>: 0E0, 5E2, 1E2, 5E3, 4E3, 1E3, 1E4, 1E5,

	1E6, 6E8 <Delivery_of_erroneous_SDUs>: 0 to 3 <Transfer_delay>: 0, 100 to 4000 <Traffic_handling_priority>: 0 to 3 Set values are saved across power cycles Values per Spec 3GPP TS 27.007
Implementation:	Fully. Setting these parameters will reset +CGQMIN and +CGQREQ to defaults.

AT+CGEREP

Description:	Control sending of unsolicited result codes
Syntax:	AT+CGSMS
Parameters:	Values per Spec 3GPP TS 27.007
Implementation:	Command accepted, no action taken

AT+CGMI

Description:	Request manufacturer identification Command processed regardless of SIM state.
Syntax:	AT+CGMI
Parameters:	
Implementation:	Fully. Init outputs “QUALCOMM™ INCORPORATED>”

AT+CGMM

Description:	Request model identification. Command processed regardless of SIM state
Syntax:	AT+CGMM
Parameters:	
Implementation:	Fully. Unit outputs: model number

AT+CGMR

Description:	Request revision identification. Command processed regardless of SIM state
Syntax:	AT+CGMR
Parameters:	
Implementation:	Fully. Unit outputs: mobile software, revision, boot block version, release date, release time.

AT+CGQMIN

Description:	Set minimum acceptable profile against the negotiated profile in Activate PDP Context Accept message
Syntax:	AT+CGQMIN=<cid>,<precedence>,<delay>,<reliability>,<peak>,<mean>
Parameters:	<cid>: 1 to 16 <precedence>: 1 to 3 <delay>: 1 to 4 <reliability>: 1 to 5 <peak>: 1 to 4 <mean>: 1 to 18, 31 Set values are saved across power cycles Values per Spec 3GPP TS 27.007
Implementation:	Fully. Setting these parameters will reset +CGEQREQ and +CGEQMIN to defaults.

AT+CGQREQ

Description:	Set the QoS Profile that is used in Activate PDP Context Request Message
Syntax:	AT+CGQREQ=<cid>,<precedence>,<delay>,<reliability>,<peak>,<mean>
Parameters:	<cid>: 1 to 16 <precedence>: 1 to 3 <delay>: 1 to 4 <reliability>: 1 to 5 <peak>: 1 to 4 <mean>: 1 to 18, 31 Set values are saved across power cycles Values per Spec 3GPP TS 27.007
Implementation:	Fully. Setting these parameters will reset +CGEQREQ and +CGEQMIN to defaults.

AT+CGREG

Description:	Presentation of unsolicited GPRS network registration status
Syntax:	AT+CGREG=[<n>]
Parameters:	<n>: 0, 1 <stat>: 0 to 5 Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CGSMS

Description:	Service Preference that will be used to send mobile-originated SMS messages.
Syntax:	AT+CGSMS
Parameters:	Parameter values supported: 2 – packet domain preferred 3 – circuit-switched preferred
Implementation:	Fully

AT+CGSN

Description:	Request product serial number identification. Command processed regardless of SIM
Syntax:	AT+CGSN
Parameters:	
Implementation:	Fully. Unit outputs: IMEI

AT+CGTFT

Description:	Traffic Flow Template
Syntax:	AT+CGTFT=[<cid>'[<packet filter identifier>,<evaluation precedence index> [,<source address and subnet mask> [,<protocol number(ipv4)/next header(ipv6)> [,<destination port range> [,<source port range> [,<ipsec security parameter index(spi)> [,<type of service(tos)(ipv4) and mask / traffic class(ipv6) and mask> [,<flow label(ipv6)>]]]]]]]]
Parameters:	<cid>: 1 to 16 <packet filter identifier>: 1, 2 <evaluation precedence index>: 0 to 255 <source address and subnet mask>: “0.0.0.0.0.0.0.0” – “255.255.255.255.255.255.255.255” <protocol number(ipv4)/next header(ipv6)>: 0 to 255 <destination port range>: “0.0” – 65535.65535” <source port range>: “0.0” – 65535.65535” <ipsec security parameter index(spi)>: “0” – “FFFFFFFF” <type of service(tos)(ipv4) and mask / traffic class(ipv6) and mask>: “0.0” – “255.255” <flow label(ipv6)>: “0” – “FFFFF” Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CHLD

Description:	Call related supplementary services
Syntax:	AT+CHLD=[<n>]
Parameters:	<n>: (0, 1, 1x, 2, 2x, 3, 4)
Implementation:	Fully.

AT+CHSN

Description:	HSCSD nontransparent call configuration
Syntax:	AT+CHSN
Parameters:	<wAiur> (in bps) 0 – TA shall calculate a proper value from currently selected network user rate (,speed> subparameter from +CBST command) 2 – 14400 4 – 28800 7 -- 57600 <wRx> 0 – TA shall calculate a proper value from currently selected <wAiur> and <codings> <topRx> 0 – TA shall calculate a proper value from currently selected <wAiru> and <codings> <codings> 0 – all supported codings are accepted
Implementation:	Fully.

AT+CHUP

Description:	Hang up voice call
Syntax:	AT+CHUP
Parameters:	
Implementation:	Fully.

AT+CIMI

Description:	Request International Mobile Subscriber Identity
Syntax:	AT+CIMI
Parameters:	
Implementation:	Fully. Unit outputs International Mobile Subscriber Identity.

AT+CLCK

Description:	Lock, unlock, or interrogate an ME or a network facility. Command is abortable.
Syntax:	AT+CLCK=<fac>, <mode>, <passwd>, <class>
Parameters:	<fac>: AB, AC, AG, AI, AO, IR, OI, OX, SC <mode>: 0 to 2 <class>: 1 to 255 Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CMEE

Description:	Report mobile equipment error
Syntax:	AT+CPBS=<n>
Parameters:	Power on value of 2 for <n> Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CMGC

Description:	Send command
Syntax:	AT+CMGC
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CMGD

Description:	Delete message
Syntax:	AT+CMGD=<index>, <deflag>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CMGF

Description:	Message format
Syntax:	AT+CMGF=<mode>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CMGL

Description:	List message
Syntax:	AT+CMGL=<stat>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CMGR

Description:	Read message
Syntax:	AT+CMGR=<index>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CMGS

Description:	Send message
Syntax:	AT+CMGS=<da>, <toda>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CMGW

Description:	Write message to memory
Syntax:	AT+CMGW=<oa/da>, <tooa/toda>, <stat>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CMMS

Description:	More Messages to Send
Syntax:	AT+CMMS=[<n>]
Parameters:	<n>: 0 -- disable 1 – enable until inter-message time expires 2 -- enable
Implementation:	Fully.

AT+CMOD

Description:	Select Call mode
Syntax:	AT+CR=<mode>
Parameters:	<mode> 0 – single mode
Implementation:	Fully.

AT+CMSS

Description:	Send message from storage
Syntax:	AT+CMSS=<index>, <da>, <toda>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CNMA

Description:	Acknowledge new message
Syntax:	AT+CNMA
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CNMI

Description:	New message indications to TE
Syntax:	AT+CNMI=<mode>, <mt>, <bm>, <ds>, <bfr>
Parameters:	<mode>: 0 to 2 <mt>: 0 to 3 <bm>: 0, 2 <ds>: 0, 2 <bfr>: 0, 1 Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CPAS

Description:	Report phone activity. Only states ready, ringing and call in progress are reported. Command processed when ME in limited service state.
Syntax:	AT+CPAS
Parameters:	Values per Spec 3GPP TS 27.007

Implementation:	Fully.

AT+CPBF

Description:	Find phonebook entries
Syntax:	AT+CPBF=<find text>
Parameters:	Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CPBR

Description:	Read phonebook entries
Syntax:	AT+CPBS=<index1>, <index2>
Parameters:	Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CPBS

Description:	Select phonebook memory storage
Syntax:	AT+CPBS=<storage>, <password>
Parameters:	<storage>: “SM”, “LD” Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CPBW

Description:	Write phonebook entry
Syntax:	AT+CPBW=<index>,<number>,<type>,<text>
Parameters:	Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CPIN

Description:	Enter PIN. Only SIM PIN/PUK and PIN2/PUK2 supported. Command processed when ME in limited service state
Syntax:	AT+CPIN=<pin>,<newpin>
Parameters:	
Implementation:	Fully.

AT+CPMS

Description:	Preferred message storage
Syntax:	AT+CPMS=<mem1>,<mem2>,<mem3>
Parameters:	<mem1>: SM, ME. MT, SR <mem2>: ME, MT, SM, SR <mem3>: ME, MT, SM, SR Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CPWD

Description:	Set new password for a facility lock function
Syntax:	AT+CPWD=<fac>, <oldpwd>, <newpwd>
Parameters:	<fac>: AB, AC, AG, AI, AO, IR, OI, OX, P2, SC Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CR

Description:	Service reporting control
Syntax:	AT+CR=<mode>
Parameters:	<mode> Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CRC

Description:	Cellular result codes
Syntax:	AT+CRC=<mode>
Parameters:	<mode> Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CREG

Description:	Presentation of unsolicitated network registration status
Syntax:	AT+CREG=[<n>]
Parameters:	<n>: 0, 1

	<stat>: 0 to 5 Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+CRLP

Description:	Alters the RLP parameters used when nontransparent circuit-switched data calls are originated.
Syntax:	AT+CRLP=<iws>,<mws>, <T1>, <N2>
Parameters:	<iws> 0 to 61 frames, for Version 0 and 1 0 to 488 frames, for Version 2 <mws> 0 to 61 frames, for Version 0 and 1 0 to 488 frames, for Version 2 <TI> 38 to 255x10ms, for Version 0 and 1 42 to 255x10ms, for Version 2 <N2> 1 to 255 retransmits
Implementation:	Fully. RLP versions 0, 1, and 2 are supported. GSM only.

AT+CSCA

Description:	Service center address
Syntax:	AT+CSCA=<sca>, <tosca>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CSCB

Description:	Selects which types of CBM's are to be received by the ME
Syntax:	AT+CSCB=[<mode>[,<mds>[,<dcs>]]]
Parameters:	
Implementation:	Fully.

AT+CSCS

Description:	Select TE character set

Syntax:	AT+CSCS=<chset>
Parameters:	<chset>: IRA, GSM, UCS2
Implementation:	Fully

AT+CSDH

Description:	Show text mode parameters
Syntax:	AT+CSDH=<show>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CSIM

Description:	Generic SIM access
Syntax:	AT+CSIM=<length>,<command>
Parameters:	<length>: 10 to 520
Implementation:	Implemented without SIM Application Toolkit support

AT+CSMP

Description:	Set text mode parameters. GSM 7-bit, 8-bit and UCS2 data coding schemes supported
Syntax:	AT+CSMP=<fo>, <vp>, <pid>, <dcs>
Parameters:	Values per Spec 3G TS 27.005
Implementation:	Fully.

AT+CSMS

Description:	Select message service
Syntax:	AT+CSMS=<service>
Parameters:	<service>: 0, 1 <bm>: not supported Values per Spec 3G TS 27.005
Implementation:	Fully. CBS commands not supported

AT+CSTA

Description:	Select type of address
Syntax:	AT+CSTA=<type>
Parameters:	<type>: 129, 145

Implementation:	Fully.

AT+CUSD

Description:	Controls unstructured supplementary service data
Syntax:	AT+CUSD=<n>, <str>, <dcs>
Parameters:	Values per Spec 3GPP TS 27.007
Implementation:	Fully.

AT+DR

Description:	Report use of V.42bis using intermediate result code before going to online data state after call answer of origination
Syntax:	AT+DR=<value>
Parameters:	<value> Values per Spec ITU-T V.25ter
Implementation:	Fully

AT+DS

Description:	Controls V.42bis data compression
Syntax:	AT+DS=<dir>,<neg>,<P1>,<P2>
Parameters:	<dir>: 1 to 3 <neg>: 0 <P1>: 512 to 2048 <P2>: 6 Values per Spec ITU-T V.25ter
Implementation:	Fully

AT+ES

Description:	Enables the Synchronous Mode
Syntax:	AT+ES=<orig_rqst>, <orig_fbk>, <ans_fbk>
Parameters:	<orig_rqst>: 6 <orig_fbk>: undefined <ans_fbk>: 1 Values per Spec ITU-T V.80ter
Implementation:	Fully.

AT+ESA

Description:	Preferred message storage
Syntax:	AT+ESA=<trans_idle>, <framed_idle>, <framed_un_ov>, <hd_auto>, <crc_type>, <nrzi_en>, <sync1>, <sync2>
Parameters:	<trans_idle>: 0 <framed_idle>: undefined <framed_un_ov>: undefined <hd_auto>: undefined <crc_type>: undefined <nrzi_en>: 0 <sync1>: 0 <sync2>: 0 to 255 Values per Spec ITU-T V.80ter
Implementation:	Fully.

AT+FAR

Description:	Adaptive Rate Control. Disable the DCE's ability to adaptively detect the selected message carrier or V.21 control message and to adjust +FRM processing accordingly.
Syntax:	AT+FAR=<value>
Parameters:	<value>: 0
Implementation:	Fully. GSM only

AT+FCL

Description:	Carrier Loss Timeout. Set the duration (<time>*100 ms) used by DEC to terminate the session if no activity is detected on the carrier, i.e. the OTA interface.
Syntax:	AT+FCL=<time>
Parameters:	<time>: 0 to 255
Implementation:	Fully. GSM only

AT+FDD

Description:	Double Escape Character. Control the DCE how to use <DLE> <SUB> pair to encode consecutive <1/0> <1/0> in data
Syntax:	AT+FDD=<value>
Parameters:	<value>: 0, 1
Implementation:	Fully. GSM only

AT+FIT

Description:	DTE Inactivity Timeout. Set the duration (in second) used by the DCE to terminate the session if the DTE fails to respond.
Syntax:	AT+FIT=<time>, <action>
Parameters:	<time>: 0 to 255 <action>: 0, 1. note: 0 and 1 are treated the same, i.e. terminate the T.31 session.
Implementation:	Fully. GSM only.

AT+FRH

Description:	Receive HDLC. Directs the DCE to receive T.30 HDLC data using the specified modulation rate (mod*100bps). Command is abortable.
Syntax:	AT+FTM=<mod>
Parameters:	<mod>: 3
Implementation:	Fully. GSM only

AT+FRM

Description:	Receive Message. Directs the DCE to receive T.30 facsimile message data using the specified modulation rate (mod*100bps). Command is abortable.
Syntax:	AT+FRM=<mod>
Parameters:	<mod>: 72, 73, 74, 96, 97, 98
Implementation:	Fully. GSM only

AT+FRS

Description:	Receive Silence. Directs the DCE to listen for silence from the remote end and report back OK when silence has been detected for the specified amount of time (in 10 ms increments). Command is abortable.
Syntax:	AT+FRS=<time>
Parameters:	<time>: 0 to 255
Implementation:	Fully. GSM only

AT+FTH

Description:	Transmit HDLC. Directs the DCE to transmit T.30 HDLC data using the specified modulation rate (mod*100bps).
Syntax:	AT+FTH=<mod>
Parameters:	<mod>: 3

Implementation:	Fully. GSM only

AT+FTM

Description:	Transmit Message. Directs the DCE to transmit T.30 facsimile message data using the specified modulation rate (mod*100bps)
Syntax:	AT+FTM=<mod>
Parameters:	<mod>: 72, 73, 74, 96, 97, 98
Implementation:	Fully. GSM only

AT+FTS

Description:	Transmit Silence. Directs the DCE to stop transmitting for the specified amount of time (in 10 ms increments)
Syntax:	AT+FTS=<time>
Parameters:	<time>: 0 to 255
Implementation:	Fully. GSM only

AT+ICF

Description:	DTE-DCE character framing This extended-format compound parameter is used to determine the local serial port start-stop (asynchronous) character framing that the DCE shall use while accepting the DTE commands and while transmitting information text and result codes to the DTE.
Syntax:	AT+ICF=<format>,<parity>
Parameters:	<format>: 3 8 data 1 stop <parity> Values per Spec ITU-T V.25ter
Implementation:	Fully, QUALCOMM™ Rm interface fixed at 8 data bits, no parity, 1 stop bit. Error returned by any other parameters.

AT+IFC

Description:	DTE-DCE local flow control. This extended-format compound parameter is used to control t he operation of local flow control between the DTE and DCE.
Syntax:	AT+IFC=<DCE by DTE>,<DTE by DCE>
Parameters:	<value> Values per Spec ITU-T V.25ter

Implementation:	Fully. Hardware and software flow control supported by Async service.

AT+IPR

Description:	Fixed DTE rate. This numeric extended-format parameter specifies the data rate at which the DCE will accept commands. Auto baud rate detection is not supported.
Syntax:	AT+IPR= <rate>
Parameters:	<rate>: 300, 600,1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400
Implementation:	Fully. Default DTE rate fixed at 115200bps.

Miscellaneous AT Commands for EU860D/EU870D

The following are miscellaneous Novatel Wireless AT commands for the EU860D/EU870D.

AT$NWDIS

Applicable Products: EU860D, EU870D

Description:

```
// Description: This command is used to emulate the HW W_DISABLE state in SW,
// such that an AND condition is implemented. If either the HW or SW switches
// indicate that we are prohibited from transmitting, we would honor that.
// This function is a mirror of the DM command to set the current operating mode
// and is used in conjunction with the W_DISABLE (where applicable) to enter
// a non-transmitting state and reset it out of that state.
```

Syntax:

```
//   | Command        | Response                        |
//   | $NWWDIS?       | $NWWDIS: HW:<hw state> SW:<sw state |
//   | $NWWDIS=?      | $NWWDIS: (0=radio xmit disabled, 1=radio xmit allowed)   |
//   | $NWWDIS=<mode> | OK, ERROR, +CME ERROR: <err>        |
//   | $NWWDIS        | ERROR, +CME ERROR: <err>            |
```

Examples:

```
// Example:     AT$NWWDIS=1  allows radio xmit
//              AT$NWWDIS=0  prohibits radio xmit
//              AT$NWWDIS?   returns current states of HW and SW signal
```

AT$NWPDN

Applicable Products: EU860D, EU870D

Description:

```
// Description: This command allows the user to power down the modem gracefully,
// while saving the MRU list, or simply save the MRU list without power down
```

Syntax:

```
// Syntax:      | Command                      | Response                       |
//              | $NWPDN?                      | ERROR, +CME ERROR: <err>       |
//              | $NWPDN=?                     | ERROR, +CME ERROR: <err>       |
// U630
//              | $NWPDN                       | OK                             |
//
// U7X0, EU7X0, U870, XU860/U870, EU870D
//              | $NWPDN                       | OK                             |
//              | $NWPDN=0                     | OK                             |
```

Examples:

```
// U630
//  AT$NWPDN
//  $NWPDN: OK            results in unit saving MRU list and powering off
//
// U7X0, EU7X0, U870, XU860/U870, EU870D
//  AT$NWPDN
//  $NWPDN: OK            does nothing, simply acceptable response
//  AT$NWPDN=0
//  $NWPDN: OK            saves the MRU list
```

AT$NWHFP

Applicable Products: EU860D, EU870D

Description:

```
// Description: HSDPA Frequency Prioritization.  Allows the user to set a
//              preferred downlink frequency.  Intended for testing purposes only
//
// Prioritization of this frequency does NOT, in any way, prevent other cells from
// being used for reselection purpose, or prevent any normal attempts to find service
// on ANY available frequency. It simply allows the user to specify the frequency to
// try first
```

Syntax:

```
// Syntax:      AT$NWHFP=0         disables frequency prioritization
//              AT$NWHFP =1,<freq>  enables frequency prioritization at specified freq
//              AT$NWHFP?          <0-1>, <freq>
//              AT$NWHFP =?
```

Examples:

```
//  AT$NWHFP=1,10700
//  $NWHFP: OK
```

CME ERROR Codes for CDMA Commands

Final result code ***+CME ERROR: <err>*** indicates an error related to mobile equipment or network, and that the command and any following commands were not executed. As no commands were executed, no result should be expected.

Table below lists and defines <err> values used by common messaging commands.

CME Error Codes

Code of <err>	Definition
0	phone failure
1	no connection to phone
2	phone-adapter link reserved
3	operation not allowed
4	operation not supported
5	PH-SIM PIN required
6	PH-FSIM PIN required
7	PH-FSIM PUK required
10	SIM not inserted
11	SIM PIN required
12	SIM PUK required
13	SIM failure
14	SIM busy
15	SIM wrong
16	incorrect password
17	SIM PIN2 required
18	SIM PUK2 required
20	memory full
21	invalid index
22	not found
23	memory failure
24	text string too long
25	invalid characters in text string
26	dial string too long
27	invalid characters in dial string
30	no network service
31	network time out
32	network not allowed - emergency calls only
40	network personalization PIN required
41	network personalization PUK required
42	network subset personalization PIN required
43	network subset personalization PUK required

44	service provider personalization PIN required
45	service provider personalization PUK required
46	corporate personalization PIN required
47	corporate personalization PUK required
100	Unknown
101...255	Reserved

CMS Error Codes for CDMA Commands

Final result code ***+CMS ERROR: <err>*** indicates an error related to mobile equipment or network. The operation is similar to ERROR result code in that none of the commands in the same command line are executed. Neither ***ERROR*** nor ***OK*** result code shall be returned.

Table below lists and defines <err> values used by common messaging commands.

CMS Error Codes

Code of <err>	Definition
0...127	GSM 04.11 Annex E-2 values, see CME ERROR codes related GSM 07.07
128...255	GSM 03.40 sub clause 9.2.3.22 values
300	ME failure
301	SMS service of ME reserved
302	operation not allowed
303	operation not supported
304	invalid PDU mode parameter
305	invalid text mode parameter
310	SIM not inserted
311	SIM PIN required
312	PH-SIM PIN required
313	SIM failure
314	SIM busy SIM wrong
315	SIM wrong
316	SIM PUK required
317	SIM PIN2 required
318	SIM PUK2 required
320	memory failure
321	invalid memory index
322	memory full
330	SMSC address unknown
331	no network service
332	network time-out
340	no +CNMA acknowledgment expected
500	unknown error
...511	other values in range 256...511 are reserved
512...	manufacturer specific
513	Unread SM on SIM

Appendix K - Data sheets

EU860D

To be provided on next release of Guide.

EU870D

To be provided on next release of Guide.

E725

To be provided on next release of Guide.

Appendix L - FAQ (Frequently Asked Questions)

In case of both EVDO and 1x signal availability does the Novatel EV620 card and/or SDK report both the signals? Or do they report only the best technology signal?

It reports only the technology that the device will be providing service on. For instance when both 1x and EVDO are available then it will report EVDO. When only 1x is available then it will report 1x.

Even if the card and/or SDK reports only the best technology signal (viz., EVDO), can a 1x connection still be initiated using the SDK, if 1x is also available?

If you are talking about packet data then the answer is no. The device will provide service on the best technology. However, with some setting changes a circuit switch call can be made on the 1x system if the carrier allows it.

When the user is connected to the network over an EVDO signal (roaming or non-roaming), and is switched to a 1x signal (roaming or non-roaming) (or vice versa), is the network connection retained? Will the Novatel EV620 SDK report the change in the signal or connection status?

If the mobile IP is used (Sprint is MIP) then the answer is yes. If not it would depend on the carrier's network layout. As for between roaming partners it will also depend on network layout but the answer is most likely no. BTW, Sprint doesn't support roaming for data service. S620 will report the change of technology and automatically switch signal reporting to the specific technology that is in use.

When the user is connected to the network over a non-roaming signal and moves to a roaming signal (or vice versa), is the network connection retained? Will the Novatel S620 SDK report the change in the roaming status?

Yes you may hand off, (hand down) from DO to 1xRTT and maintain the connection (although you must be in dormant mode). You can not hand up from 1xRTT to EV-DO.

1xRTT hand down can occur when EVDO is in either active or dormant mode, there is no limitation. 1xRTT to EVDO hand up on the other hand has to wait until 1xRTT is in dormant mode before it can happen. Thus if you are continuously active on TCH in 1xRTT you will not hand up to EVDO even when you are back in EVDO coverage.

Will the Novatel EV720 SDK report the change in the signal or connection status?

Yes. SDK will report change of status.

Appendix M - Glossary

Abbreviations given in 3GPP TR 21.905 [15] and the following] apply.

Access Point Name (APN)
The IP domain name (i.e. Novatel Wireless.com) of the network device that acts as a gateway by connecting a CDMA wireless radio network to a wired local or wide area network.

Active Network Session
An active network session allows you to send and receive data across the Internet using point-to-point protocol through your network connection.

Anonymous Access (AA)
Network does not know the real identity of the mobile. Opposite to non-anonymous.

AP Access Point
An entry point to an external network.

AT Commands
AT commands are a language type that enables PC communications software to give the modem directions. The term ***AT*** comes from the command terminology which always begins with ***attention***, or AT.

Authentication Authorization Accounting (AAA)
Used as shared secret passwords during a Mobile IP registration.

Baud Rate
The actual bit rate, excluding compression and other TX enhancements, on a communication line.

Border Gateway (BG)
Logical box that connects two (or more) operators together via an Inter-PLMN backbone. BG protects operator’s intra-PLMN network against intruders.

Carrier

See Service Provider

Circuit Switched Data
A wireless network connection established, using a single circuit that extends from you, directly through the network to your call’s destination. Opposite to packet switched.

CLI
Command Line Interface.

CLIR
Call Line Identification Restriction.

Code Division Multiple Access (CDMA)
Code Division Multiple Access is a spread spectrum wireless access technology that allows multiple users to share the same physical RF channel (1.25MHz for single carrier direct spread 1X) by use of orthogonal code spreading.

Connection Oriented Network Service (CONS)
Same as X.25 protocol for packet network transmission and switching.

Connection Profile

See Network Connection Profiles

CSD

See Circuit Switched Data

dBm
dBm stands for decibels below 1 milliwatt. It is essentially a device's signal output power compared against a standard input signal strength of 1 (one) milliwatt. This number represents a ratio and is expressed as a negative number (i.e. -60dBm)

Default Network Connection Profile
The default network connection profile is the connection profile, chosen by you that the Modem Manager will use to connect to the network. The default network connection profile can be thought of as the ***active*** network connection profile.

Direct Memory Access (DMA)
A fast method of moving data from a storage device or LAN device interface card directly to RAM which speeds up processing. DMA by passes the CPU.

Domain Name
The name assigned to a computer or group of computers that constitute an IP network domain. In general, a domain name is comprised of its local host name and its top-level domain. The top-level domain can be made up of several names, each separated by a period (.).
An example of a domain name is ***novatelwireless.com***.

Domain Name System (DNS)
This is a network server used on IP networks, such as the Internet, for translating network host names and Universal Resource Locators (URL's) into IP addresses.

Domain Name System (DNS) Address
The IP (internet protocol) address of the Domain Name System (DNS).

Edge Technologies
Edge (Enhanced Data for Global Evolution) is an enhanced version of GPRS providing three to four times more capacity and data throughput. Average speeds range from 100 to 130 kbps with theoretical peak data rates of 473 kbps. Average rates are fast enough to support a wide range of advanced data services such as streaming audio and video, fast Internet access and large file downloads, EDGE can also support a greater range of enterprise applications, and more multimedia applications including push-to-talk services.

EVPF
Enhanced Validity Period Format

Firmware
Firmware is a program or set of programs that have been set permanently into a computer chip. The programs themselves usually are low-level programs that directly manipulate or interact with the hardware. An example of firmware is your desktop computer's BIOS.

General Packet Radio Service (GPRS)
GRPS is a packet-based, always-on data connection standard.

High Speed Downlink Packet Access (HSDPA)
HSDPA is a packet-based data service that improves upon UMTS by increasing speeds to 500-800 kbps with peak data rates of up to 10 Mbps (five times faster than UMTS and other 3G technologies) in a 5 Mhz channel. In addition, HSDPA significantly improves packet data throughput capacity, thereby increasing the number of users that can be supported at higher data rates on a single radio carrier.

Home Agent (HA)
A router in the home subnet of the mobile node. Used in part with Mobile IP.

Home Public Land Mobile Network (HPLMN)
The home network.

IMEI
IMEI stands for International Mobile Station Equipment Identity.

Internet Protocol (IP)
Internet Protocol works in conjunction with Transmission Control Protocol (TCP). TCP/IP are part of a group of protocols that provide communication across interconnected networks. TCP/IP is the protocol used on the Internet. The TCP protocol first establishes a connection between the two systems in order to send and receive data, and then breaks and sequentially marks the message into small packets. The IP protocol routes and sends the packets based on the IP address.

Internet Control Message Protocol (ICMP)
IP network control protocol.

Internet Over-The-Air (IOTA)
Network operators can remotely provision a device on their network by using an Internet Over-The-Air implementation. Usually, a newly purchased device needs to initiate an IOTA session to perform provisioning before it is allowed to be on the operator's wireless network.

IP Addresses
As with personal computers that access the Internet, modems using CDMA technology also have a dedicated Internet Protocol (IP) address, which is used to identify the node or access point for the modem on the Internet. The service provider assigns this IP address.
The 32-bit host address is usually represented in dotted decimal notation, e.g. 128.121.4.5. The address can be split into a network number (or network address) and a host number unique to each host on the network and sometimes also a subnet address.

IP Network
A network of computer networks that employ Internet Protocol allowing a user to access the Internet, provided that the user has a modem; telephone line, cable line, or wireless data network (e.g. CDPD); and a service provider.

Local Area Network (LAN)
A computer network that spans a relatively small area (typically up to a 1 km radius), although most LANS are confined to a single building or group of buildings. This type of networking allows for easy interconnection of terminals, printers, and computers within a building or buildings.

Logical Link Control (LLC)
Protocol layer between MS and SGSN.

Medium Access Control (MAC)
Protocol in the radio level that is used to allocate the radio channel.

Megahertz (MHz)
One million hertz. Hertz is another word for cycles in a radio frequency.

Mobile Directory Number (MDN)

Mobile Identification Number (MIN)

Mobile IP (MIP)
Mobile IP provides a method to allow IP traffic to find nodes whose point of attachment to the Internet changes.

Mobile Node (MN)
This is either an application running on a handset, or a data device connected to a handset with IP connectivity.

Mobile Station (MS)
The device being used to receive/transmit data and/or voice.

Mobile-Terminal (MT)

Network Access Identifier (NAI)
Used as an Identifier/login for Mobile IP.

Network Connection Profiles
A network connection profile is a group of connection settings that define a specific network connection to the Modem Manager. This includes such settings as network ID, network password, APN, DNS addresses and so on.

Novatel Wireless Inc. (NWI) (NVTL)

Original Equipment Manufacturer (OEM)
The original manufacturer of a pieces of equipment, typically complete boards, duplexers or enclosures etc.

Over-The-Air (OTA)

Packet
A short block of data transmitted across a network.

Packet Data Network (PDN)
Network that carries user data in packets. ex. Internet and X.25

Packet Size
The size of a packet expressed in bytes.

PC Card
A PC Card, similar in size to a credit card, is used for adding devices on to portable computing devices such as laptop, handheld, and palmtop computers. Some examples of these devices are modems, network cards, disk drive adapters, and extra memory. PC cards are often called PCMCIA cards.

Peripheral Component Interconnect Special Interest Group (PCI SIG)

PCMCIA Card Slots
The PCMCIA card slots are the sockets in the computing device, in which the PCMCIA card is inserted. It is the hardware interface between the computing device and the PCMCIA card.

Point to Multipoint (PTM)
Widely used IP protocol used to connect, i.e. PC and ISP via modems.

Point to Point (PTP)
One sender on receiver.

Point-to-Point Protocol (PPP)
PPP is an interconnection protocol which allows a device, such as a wireless IP modem, to connect to a network or the Internet.

Primary Domain Name System
In order to get the translated IP addresses, the modem will try to connect to the server with the primary DNS address. If the modem cannot connect to this address, it will try to connect using the secondary DNS address.

Primary Roaming List (PRL)

Protocol Data Unit (PDU)
One data packet.

Quality of Service (QoS)
Definition of the service class of the connection between MS and the network.

Radio Link Protocol (RLP)

Registration
In order to send and receive data across a given network, a CDMA modem must first register to a CDMA network. This involves the selection of an available channel and interaction with various systems on the CDMA network to set up a communication path.

Remote Access Service (RAS)
Software that enables distant PCs and workstations to get into a Remote Access Server to retrieve software and/or data on a corporate LAN. This service is provided through modems, analog telephone lines or digital ISDN lines.

Routing Area (RA)
A set of cells that belongs to one group. RA is always a subset of an LA (Location Area).

Secondary Domain Name System
If the modem cannot connect to the DNS using the primary address, it will try to connect using the secondary DNS address.

Security Parameter Index (SIP)
Used in part with Mobile IP.

Segment
Each IP network address consists of four numeric segments, which are divided by a period ("."). For example, 204.119.63.40.

Service Provider
A company that provides network connections to the Internet.

Short Message Service (SMS)
Short messages either in binary (160 characters) or text messages (140 bytes) format.

Terminal Equipment (TE)

Transmission Control Protocol (TCP)
Protocol layer on top of conventional IP protocol.

Type II PCMCIA Card
A Type II PCMCIA card is identical to the Type I PCMCIA card in all ways except that it is thicker than the Type I card. The Type II PCMCIA card is in general use now.

U_m
Mobile-to-Base Station air interface link.

Universal Product Support Tool (UPST)
The Universal Product Support Tool (UPST) consists of the UPST Framework and UPST device DLLs. The UPST Framework is a Windows 32 application (UPST.exe) that uses UPST device Dynamic Link Libraries (DLLs) to provide basic device provisioning functions such as Refurbish, Software upgrades, Preferred Roaming List (PRL) upgrades, and Phone Settings programming.

User Datagram Protocol (UDP)
Another protocol on top of IP.

Wireless IP Network
A wireless network (e.g. CDMA) that uses Internet Protocol (IP)