

AT&T Developer Program

AT&T Network Services White Paper

Best Practices and Case Studies

Document Number **NSG_WP_01**
Revision **1.0.1**
Revision Date **10/06/10**



Legal Disclaimer

This document and the information contained herein (collectively, the "**Information**") is provided to you (both the individual receiving this document and any legal entity on behalf of which such individual is acting) ("**You**" and "**Your**") by AT&T, on behalf of itself and its affiliates ("**AT&T**") for informational purposes only. AT&T is providing the Information to You because AT&T believes the Information may be useful to You. The Information is provided to You 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 AT&T has exercised reasonable care in providing the Information to You, AT&T does not warrant the accuracy of the Information and is not responsible for any damages arising from Your use of or reliance upon the Information. You further understand and agree that AT&T in no way represents, and You in no way rely on a belief, that AT&T is providing the Information in accordance with any standard or service (routine, customary or otherwise) related to the consulting, services, hardware or software industries.

AT&T DOES NOT WARRANT THAT THE INFORMATION IS ERROR-FREE. AT&T IS PROVIDING THE INFORMATION TO YOU "AS IS" AND "WITH ALL FAULTS." AT&T DOES NOT WARRANT, BY VIRTUE OF THIS DOCUMENT, OR BY ANY COURSE OF PERFORMANCE, COURSE OF DEALING, USAGE OF TRADE OR ANY COLLATERAL DOCUMENT HEREUNDER OR OTHERWISE, AND HEREBY EXPRESSLY DISCLAIMS, ANY REPRESENTATION OR WARRANTY OF ANY KIND WITH RESPECT TO THE INFORMATION, INCLUDING, WITHOUT LIMITATION, ANY REPRESENTATION OR WARRANTY OF DESIGN, PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, OR ANY REPRESENTATION OR WARRANTY THAT THE INFORMATION IS APPLICABLE TO OR INTEROPERABLE WITH ANY SYSTEM, DATA, HARDWARE OR SOFTWARE OF ANY KIND. AT&T DISCLAIMS AND IN NO EVENT SHALL BE LIABLE FOR ANY LOSSES OR DAMAGES OF ANY KIND, WHETHER DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, SPECIAL OR EXEMPLARY, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF GOODWILL, COVER, TORTIOUS CONDUCT OR OTHER PECUNIARY LOSS, ARISING OUT OF OR IN ANY WAY RELATED TO THE PROVISION, NON-PROVISION, USE OR NON-USE OF THE INFORMATION, EVEN IF AT&T HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES OR DAMAGES.

AT&T Proprietary

The information contained here is for use by authorized person only and is not for general distribution

Revision History

© 2010 AT&T Intellectual Property
All rights reserved.

AT&T and AT&T logos are trademarks of AT&T Intellectual Property.

All marks, trademarks, and product names used in this document are the property of their respective owners.

Date	Revision	Description
08/25/10	1.0	Initial version focused on SOAP / Parlay X Web services.
10/06/10	1.0.1	Copyedits and reference updates for AT&T Developer Program website.

Table of Contents

- 1. Introduction 1
 - 1.1 Audience 1
 - 1.2 Contact information 1
 - 1.3 Resources 1
 - 1.3.1 AT&T resources 1
 - 1.3.2 Other resources 2
 - 1.4 Terms and acronyms 2
- 2. Overview 4
- 3. Network services APIs 4
 - 3.1 Developer learning curve 5
 - 3.2 Network and device aware application development 5
 - 3.3 Improve usability and customer experience 6
 - 3.4 AT&T APIs: today 6
 - 3.5 AT&T APIs: future 7
 - 3.6 AT&T network services application patterns 7
- 4. Top 10 best practices for network service development 11
 - 4.1 Development and deployment 11
 - 4.2 Prepare for AT&T lab certification 12
 - 4.3 Utilize application server to centralize business logic 12
 - 4.4 Offload CPU-intensive tasks to a server 13
 - 4.5 Cache responses on application server to minimize API calls 13
 - 4.6 Use standard XML parsing methods 13
 - 4.7 Consider leveraging a cloud solution to provide scalability 14
 - 4.8 If status is required, check within a time constraint 14
 - 4.9 Utilize threads and connection pools to process requests 14
 - 4.10 Build a device database or libraries to store device information 15
- 5. AT&T developer support for using network services 16
 - 5.1 Developer registration 16
 - 5.2 Network service information sources 16
 - 5.3 Application provisioning for access to network services 16
 - 5.4 Application certification 17
 - 5.5 Production launch 17
- 6. Sample use cases for AT&T Network Services 18

AT&T Proprietary
 The information contained here is for use by authorized
 person only and is not for general distribution

Table of Contents

6.1	Successful application originated transaction.....	18
6.2	Changing the service address sample code.....	19
6.3	Authentication and authorization	20
6.4	Authentication and authorization sample code.....	21
7.	Security.....	23
7.1	HTML escape data persisted or sent through network services	23
7.2	Error auditing	23
7.3	Explicitly control size of data being sent or received.....	23
7.4	Protect against XML schema validate attacks.....	24
7.5	Limit feature access.....	24
8.	Tools.....	25
8.1	Development tools	25
8.2	Testing tools	25
8.2.1	Quality management tool use	26
8.2.2	Test from all points of view	26
8.2.3	Evaluate categories	26

Figures

Figure 1.	Currently supported and future supported deployment models	11
Figure 2.	Successful Application Originated–Connection Established	19
Figure 3.	Successful Application Originated–Connection Established	23

Tables

Table 1.	Terms and acronyms.....	2
Table 2.	Application design and deployment patterns.....	8

1. Introduction

This white paper provides you the best practices to develop applications using network services APIs as part of the AT&T Developer Program.

1.1 Audience

We created this white paper for technical specialists involved in developing applications that would benefit from using AT&T network services APIs. This includes business and technical architects, software developers, and testers responsible for designing and building applications that use the AT&T Network Services Gateway. To make the best use of the document, we recommend you have working knowledge of the following:

- XML, SOAP, Web services, and REST.
- Java or .Net based programming languages.
- One or more server-side technologies, such PHP, Perl, Python, Ruby, or ASP.net.
- Client-side interactive technologies, such as JavaScript, AJAX, or widget frameworks.

1.2 Contact information

E-mail any comments or questions regarding this document to developer.program@att.com. Please reference the title of this document in your e-mail.

1.3 Resources

1.3.1 AT&T resources

The AT&T Developer Program website, <http://developer.att.com>

- Network Services Tech Tips
- Application Development Best Practices white paper

- Mobile Application Development Best Practice white paper
- Emerging Mobile Application Architectures white paper
- Mobile Application Development quick start guide

1.3.2 Other resources

World Wide Web Consortium website:

- Mobile Web Application Best Practices, <http://www.w3.org/TR/mwabp>

Accenture website:

- Three Screen strategy for application development white paper, http://www.accenture.com/Global/About_Accenture/GCF/Three_Screens.htm

1.4 Terms and acronyms

The following table defines terms and acronyms used in this document.

Table 1. Terms and acronyms

Term or Acronym	Definition
API	Application programming interface
BI	Business intelligence
CORBA	Common object request broker architecture
DCS	Device capability services
DOM	Document object model
ESB	Enterprise service bus
GPS	Global positioning system
HTTP	Hypertext transfer protocol
IDL	Interface definition language
IM	Instant messaging
IPTV	Internet protocol television
JAXB	Java architecture for XML binding
JAXP	Java API for XML processing
JAXR	Java API for XML registries
JSON	JavaScript object notation
MIME	Multipurpose internet mail extensions
MMS	Multimedia messaging service

Term or Acronym	Definition
PHP	Hypertext preprocessor [recursive acronym]
REST	Representational state transfer
RPC	Remote procedure call
SAAJ	SOAP with attachments API for Java
SDP	Service delivery platform
SLA	Service level agreement
SMS	Short messaging service
SOAP	Simple object access protocol
SSL	Secure socket layer
SSO	Single sign on
TLS	Terminal location service
TPS	Transactions per second
UAP	User agent profile
URL	Uniform resource locator
WSDL	Web service description language
WAP	Wireless access protocol
XHTML	Extensible hypertext markup language

2. Overview

Current market trends suggest a one-sided view on application developer platforms. There is a focus on application development centering on a specific mobile device. This compares to centering development on network resources. Developers interested in taking advantage of the market trends in connected devices focus on network-centric development.

We feel an important aspect of application development is to deploy network capabilities intelligently across various platforms. This includes mobile devices, personal computers, and IPTV. Our network services APIs meet the needs applications designed to work across various platforms.

We believe you should have options to design and implement applications with business components and logic split or spread across multiple devices and networks. This lets you implement applications where most of the business logic resides on the network server or the application server. You can easily roll out thin clients on various platforms at the same time, and with less turnaround time.

To use network capabilities intelligently, you need to know the use cases for when device centric applications or device agnostic applications can work better on your target devices. If you make an informed decision on the device an end consumer uses, you can make choices of rendering the content effectively to the end consumer. This lets you develop applications that work seamlessly across devices and use the same APIs.

3. Network services APIs

Network services APIs are not bound to a particular software system. They are available remotely over the network. Web services (WSDL and SOAP based) and REST are specific API models. They use XML or other lightweight formats for standard messages, like JSON. The exchange is with a protocol like HTTP.

Developers routinely use GPS chips, accelerometers, cameras, compasses, and other high-tech widgets embedded in smart-phones. More often than not, developers fail to take advantage of rich mobile network capabilities.

Applications that run server-side business logic implemented an application server can use AT&T network services, even when device capabilities are not available.

You can use unique network capabilities, such as the location of a handset, network characteristics such as the connection speed, and messaging facilities like SMS and MMS. This enhances an application to provide a richer user experience. Providing strong authentication mechanisms and payment services using two-way SSL, you can use AT&T services to drive greater revenue.

We make network capabilities like SMS, MMS, and terminal location available to developers in standards-based Web services (and in the future, REST services). This enables you to develop applications in an easy to use and globally accepted manner.

Here are some compelling factors to motivate you to start harnessing AT&T network services APIs in your applications.

3.1 Developer learning curve

Applications and networks currently work largely independent of each other. This hinders you from using network interfaces because network operators typically use different APIs. This means you may have to adapt your application for each network and platform so it can use the application.

We see providing standards-based AT&T Network Services APIs. This includes Parlay X. You can adapt your applications across various networks and platforms easily.

We are in the process of standardizing AT&T network services APIs through the OneAPI initiative. These efforts aim to provide you with a common, lightweight, and web-friendly API that many different mobile operators support. This minimizes specific device or platform learning requirements. This standardizes the use of network services APIs across various devices, platforms, and networks.

3.2 Network and device aware application development

AT&T APIs are based on proposed and adopted open standards for network services APIs. These insulate developers from monitoring and changing carrier network configurations by eliminating the need to build applications that use native and/or proprietary protocols to communicate with the network elements.

Network monitoring can be independent of applications. Adaptive approaches typically are highly dependent on applications. Applications that capitalize on

wireless features, mobility, and multimedia content all pose challenges to network awareness and network adaptation.

AT&T APIs shield you from some application development complexities by hiding network details. You do not need to know the under-the-hood network details in order to build your application. This includes active-passive monitoring and passive monitoring to derive accurate information from network.

AT&T network services APIs help you to harness network awareness states. They also help you to design and implement rich applications.

3.3 Improve usability and customer experience

You have an option of designing applications that give consumers more control over how they experience their content. Today consumers want a compelling experience across their three primary view screens: mobile device, personal computer, and television (IPTV). The AT&T network services APIs provide an important option for you to develop applications that work across all three screens seamlessly.

3.4 AT&T APIs: today

We have deployed a multi-tiered infrastructure to help you to harness AT&T network capabilities in effective and efficient manner.

The AT&T Network Services Gateway supports following services network services API for you to use to develop applications:

- Short Messaging Service (SMS)
- Multimedia Messaging Service (MMS)
- Wireless Access Protocol (WAP) Push Service
- Terminal Location (TL) Service
- Device Capabilities (DC) Service

For additional information regarding AT&T-supported APIs, visit the official website of the AT&T Developer Program at <http://developer.att.com>.

Today, access to AT&T APIs requires server-side business logic. You have an option to develop and deploy server-side application business logic on the

“cloud,” third party facilities, or any enterprise data center. This can be in part or in whole. The server-side components access AT&T network services using Web service API calls and act as a broker for the API responses. An appropriate agent—i.e. a client application, widget, or portlet— designed to interact with your server-based application runs on the desired target devices.

3.5 AT&T APIs: future

In the future, AT&T network services will allow you to design and implement applications with business logic split across distributed components (various computing devices). Future AT&T APIs will allow client-side (agent-side) business logic or server-side business logic.

AT&T client APIs will let you design and implement applications on the client by using APIs as a Web service or as a REST call. You will be able to do this in part or in whole. Availability will be on mobile devices, browsers, personal computers, native platforms, and gaming devices.

The AT&T Developer Program roadmap focuses on implementing network services APIs that enable you to develop and deploy applications across multiple platforms and connected devices. You can extend the reach of applications that implement network services awareness beyond PCs and mobile devices, to include compatible set top boxes, IPTVs, and gaming devices.

3.6 AT&T network services application patterns

Distribution of business logic across the client or server allows creation of very innovative application patterns. Table 2 on the next page summarizes these application patterns.

Table 2. Application design and deployment patterns

Pattern	Characteristics	Real life scenarios
<p>Pattern 1: Client only</p>	<p>Characteristics include the following:</p> <ul style="list-style-type: none"> • Mobile devices. Devices include Java, Brew, BlackBerry, WinMobile, iPhone, and Android among others. • Browser or widget platform. Examples include using JavaScript. • Native apps. Platforms include Windows, Linux, various gaming devices, and set top box among others. 	<p>Implementations include:</p> <ul style="list-style-type: none"> • Tetris running on a mobile device. • Xbox single player game.
<p>Pattern 2: Client with network services awareness</p>	<p>Characteristic include the following:</p> <ul style="list-style-type: none"> • Mobile devices. Devices include Java, Brew, BlackBerry, WinMobile, iPhone, and Android among others. • Browser or widget platform. Examples include using JavaScript. • Native apps. Platforms include Windows, Linux, various gaming devices, and set top box among others. • Client using Network Services. Any of the above clients making direct REST or Web service request of one or more network services. 	<p>Implementations include:</p> <ul style="list-style-type: none"> • Provide weather information using a Web widget running in several contexts using the user's data when that by making a location-based service (LBS) request. • Send and receive SMS messages while using a lean back-TV application that does a lookup of contacts from AT&T Address Book (AAB) with presence so the user can send an SMS relating to current program.

Pattern	Characteristics	Real life scenarios
<p>Pattern 3: Client and server with network services awareness</p>	<p>Client makes a request of the server for some capability. The server brokers the requests one or more network services and provides the response to the client.</p>	<p>Implementations include:</p> <ul style="list-style-type: none"> • Digital video recording (DVR) application requests the server to check if any friends like the same category of program and selectively sends those friends a message to their universal message box suggesting that they should record program also. • If policy allows a subscriber to schedule a program on behalf of a friend, the DVR schedule notifies both parties of the event.
<p>Pattern 4: Server with network services awareness</p>	<p>Characteristic include the following:</p> <ul style="list-style-type: none"> • The client has the most basic of capabilities, like HTML 3 forms (browser or widget) or streaming media player. • The server uses application business logic, using one or more network services. 	<p>Implementations include:</p> <ul style="list-style-type: none"> • Similar to pattern 3, but the agent relies on nothing more than HTML 3 forms and basic markup to render results. • Basic store locator website that renders hours and driving instructions to nearest store, based on location of the subscriber whether sitting in front of an IPTV at home or location of a mobile device.

Pattern	Characteristics	Real life scenarios
Pattern 5: Server only	Conventional Web server application, relying on minimum support from the Web, client, or a browser or widget.	Implementations include: <ul style="list-style-type: none"> • Simple small business website. • Personal website with basic blogging capabilities.

Note. You can consider thick or thin client deployment options where there is mention of a client pattern in the above table. Whenever feasible, a thin client is recommended for mobile applications to minimize the data volume transferred over legacy wireless networks such as 2G and EDGE.

From Table 2, a client can utilize an API directly for something like AT&T Maps Services. A server can utilize an API directly for something like a digital video recorder server responding to a client SMS and scheduling programs. There could be cases when the client is a dumb terminal or supports minimal functionality and the server handles all network and business logic, such as rendering of services on televisions. There could also be a pure server centric deployment, where an enterprise server may access AT&T network services to deliver recommendations. You have to choose where to use the API—the client or server.

4. Top 10 best practices for network service development

4.1 Development and deployment

We recommend you choose application development and deployment patterns listed in section 3.6 to arrange business logic efficiently between server and agent to best capitalize on computing horsepower, latency, and network capabilities.

Split the business logic based on a real-life scenario or use case for the application that is being realized.

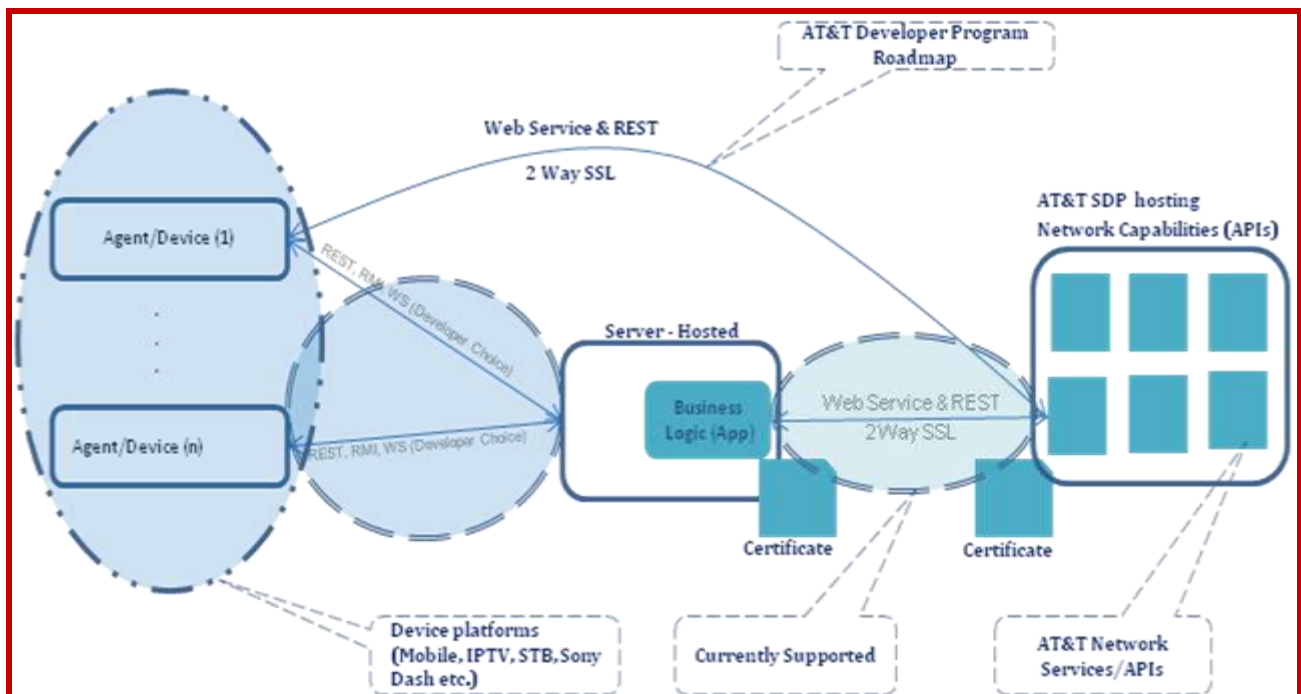


Figure 1. Currently supported and future supported deployment models

IMPORTANT: The AT&T Developer Program currently supports an application development and deployment model where you implement server-side application logic using AT&T APIs (patterns 3 and 4). Clients will be able to interface with the server using various Web technologies, such as RMI, CORBA, Web Services and (in the near future) REST. Our roadmap includes support for application deployment pattern #2 using AT&T network services.

AT&T Proprietary

The information contained here is for use by authorized person only and is not for general distribution

4.2 Prepare for AT&T lab certification

If you are building an AT&T co-branded application, using an AT&T branded distribution channel, or require high transaction volumes for your application, AT&T certification of your application will be required before it is deployed to our production network.

We recommend you perform comprehensive end-to-end testing of the application using the [AT&T Sandbox](#) environment prior to submitting the application to the AT&T certification process.

At a high level, aim to complete testing in following categories:

- A. Functionality assessment.
- B. Interoperability assessment.
- C. Usability assessment.
- D. Performance assessment.
- E. Exception handling assessment.

The AT&T certification process verifies your application complies with AT&T standards. Quality engineering and user acceptance testing are outside the scope of the certification process.

4.3 Utilize application server to centralize business logic

While using the AT&T network services APIs, especially Web services based on WSDL (SOAP), design you application in such a way that network latency and heavy XML document handling have no adverse implications on the performance of the application.

- Perform heavy business logic processing and parsing of deep-nested XML on the application server.
- Simplify (flatten) XML used in data exchange with the application client.

While addressing performance issues, focus on the Agent to Application Server link.



4.4 Offload CPU-intensive tasks to a server

To optimize performance and scalability, consider using the following techniques:

- Design your application server with a scalable, connector-based architecture to allow easy addition of new network services APIs and application clients.
- Consider whether automatic scaling or connection pooling can improve performance.

4.5 Cache responses on application server to minimize API calls

By implementing a cache scheme, you can prevent generating a policy exception response from the API and you can minimize your cost to access the API.

For example, if your application is provisioned for the Terminal Location Web service at one transaction per second, and the same user clicks a button that causes the getLocation request to occur 3 times a second for 5 seconds, your application will receive a policy exception. The application server does not need to make 15 requests over a 5-second period. The server logic can be designed to cache the other 14 client requests while the first request is being processed.

4.6 Use standard XML parsing methods

We recommend that you do the following:

- Avoid using custom parsers to prevent loopholes.
- Parse messages using SAX to avoid a denial of service attack through inputting of large XML files.
- If using DOM parsing, set a maximum limit on message size.

Note. If you use a scripting environment like Perl or Ruby, several parsers are available. Java has several options available. Do your homework. It can save you coding and ensure the best performance possible at the same time.

4.7 Consider leveraging a cloud solution to provide scalability

We recommend that you investigate using cloud services and cloud computing for your application. Deploying applications on the cloud provides benefits such as high availability and redundancy, lower capital and operating costs, and the ability to scale capacity to meet demands for growth quickly.

4.8 If status is required, check within a time constraint

We recommend that you design server-based applications that use network services with configurable time parameters for message retrieval, status polling, and message notifications.

For example, if an application that supports receiving SMS messages from the NSG must invoke this request at least once every 30 minutes. Otherwise, NSG may automatically remove the messages.

For applications that expect to receive many SMS messages, we recommend that the application invoke this request more frequently, but with a maximum frequency of every 5 seconds. Depending on the volume of processed messages, you may want to change the time parameters to optimize the retrieveSms retrieval frequency.

4.9 Utilize threads and connection pools to process requests

While using AT&T APIs, we recommend that you consider using connection pooling where applicable.

Designing your server-side application with the capability to maintain a connection pool or pools enables the application to perform optimally when the network resources are at a premium. Depending on the round trip delay, you can choose to use the connections from an existing pool rather than create a new connection by invoking the API.

4.10 Build a device database or libraries to store device information

We recommend that you design your server-based applications to serve a mobile device with device libraries or a device database. This stores device capabilities information for your application to use to deliver the optimal user experience to device users.

There are multiple sources for device information on the market today, including:

- User-agent string on the device.
- Manufacturer-provided UAProf URLs.
- User-agent header captured from the network gateway.
- AT&T Device Capabilities Web service.
- The AT&T Developer Program device database, located in the Devices section of the website.
- Device Anywhere.
- WURFL.

To optimize device information accuracy and improve the mobile device user's experience of your application, consider using the following techniques:

- Implement server-side logic that stores the highest quality results of device information in the device database.
- Design the application server to obtain and compare device information from multiple sources if your device database does not already contain a device's profile.
- When feasible, use the information from your device database in the application server logic for invoking an applicable network services request.

5. AT&T developer support for using network services

We encourage you to innovate with network services. Here is an overview of AT&T developer support for network services to help you develop applications. The AT&T Developer Program [website](#) provides additional information about the process. Review the information available in the **Develop > AT&T Innovation Centers & Network Services** section of the website for further details.

5.1 Developer registration

You must join the AT&T Developer Program to create applications using AT&T network services. Participation in the program makes communication easier between AT&T technical teams and you.

5.2 Network service information sources

The AT&T Developer Program website offers tools and resources. This includes white papers, programming guides, and sample code for developers. This provides a head start in developing applications. You need a user name and password from your developer registration to access this content on the website.

5.3 Application provisioning for access to network services

Provisioning of an application for network services access has three parts:

1. **Service specific parameters.** For example, if you are using SMS MO or MT, use short code for MO.
2. **Authentication and SLA parameters.** For example, specify things like SSL cert registration, user and password, and quota values.
3. **Network access if *not* using SSL over public Internet.** For example, are you using VPN or AVPN over the public Internet?

The AT&T Sandbox uses service “bundles” to automatically provision a set of available network services and application credentials for you to use during testing. You must choose the specific services you plan to utilize you’re your application is on-boarded to the production environment.

5.4 Application certification

Depending upon the specific SLA and your distribution channels (such as AT&T branding, or co-marketing), your application may require certification.

5.5 Production launch

An application achieves production launch readiness at the completion of the following actions:

- The developer signs the AT&T-approved legal contact with terms and conditions.
- The developer and AT&T agree to financial terms, if applicable.
- Provisioning of the application on the production network.

The AT&T Developer Program provides you support to achieve this goal.

6. Sample use cases for AT&T Network Services

Here are some samples on how you can use the Web services version of APIs on AT&T Network Services. For a complete set of service-specific call flows, refer to the *AT&T Network Services API Developer Guide*.

For details on support of native network services APIs, refer to the **Network Services API Catalog** on AT&T Developer Program website at <http://developer.att.com>.

These use cases apply across the network capabilities supported by the AT&T Developer Program. For the behavior of specific APIs, refer to the *Network Services API Developer Guide*.

6.1 Successful application originated transaction

This sequence diagram shows a third party application establishing a connection with a Network Services Gateway (NSG) end point. The sequence may be summarized as follows:

1. A third party application initiates an SSL connection with AT&T NSG.
2. NSG and application negotiate an authentication algorithm.
3. NSG and application exchange certificates.
4. NSG validates the client's certificate and establishes an SSL connection.
5. NSG and application may exchange a Web service request and response using call flows documented in the *Network Services API Developer Guide*. The request from the application must contain a SOAP header that includes the WS-Security token containing the user name.

Note. If the connection established during steps 1-4 is a persistent connection, the connection remains open for later requests until one of the following occur:

- The third party or NSG make a request to close the connection.
- The processing of specified number of transactions (request or responses) of any type finishes.
- The connection has been idle for few seconds.

- An error occurs.

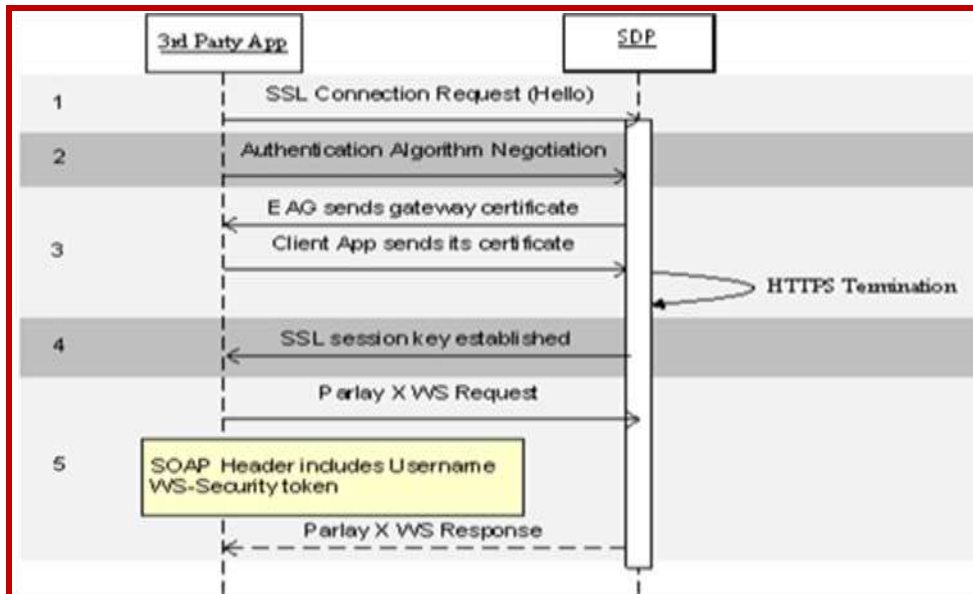


Figure 2. Successful Application Originated–Connection Established

6.2 Changing the service address sample code

When an application uses a Web service, it is common to change the address (URL) used reach the service. The address used is based upon the specified soap:address within the service’s WSDL file. For example, the wsimport utility uses the address from the WSDL in the code generated.

This code sample illustrates how the WSDL for the Device Capabilities Service may define the soap:address:

```
<wsdl:service name="DeviceCapabilitiesService">
  <wsdl:port name="DeviceCapabilities" binding="tns:DeviceCapabilitiesBinding">
    <soap:address
location="https://sampleSDPhost:9080/DeviceCapabilitiesService/services/
DeviceCapabilities"/>
  </wsdl:port>
</wsdl:service>
```

This code sample illustrates how you can modify the address (overriding the value in the WSDL) from within the application code:

```
DeviceCapabilitiesService service = new DeviceCapabilitiesService();
DeviceCapabilities port = service.getDeviceCapabilities();
BindingProvider bp = (BindingProvider) port;
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
"https://SDP/pxdc/services/DeviceCapabilitiesService");
```

6.3 Authentication and authorization

Use the WS-Security (WSSE) UsernameToken for the third party application authentication to SDP.

This shows the format of the UsernameToken. The WSSE User Name Token Profile 1.1 defines this.

```
<wsse:UsernameToken wsu:Id="Example-1">
  <wsse:Username> ... </wsse:Username>
  <wsse:Password Type="..."> ... </wsse:Password>
  <wsse:Nonce EncodingType="..."> ... </wsse:Nonce>
  <wsu:Created> ... </wsu:Created>
</wsse:UsernameToken>
```

Where,

- *Username* contains the third-party application authentication ID in the following format: `UserName:GroupName`
- *Password Type* will be `PasswordText` and contains the password in clear text.
- *Nonce* contains a randomly generated number that should be changed for each request.
- *Created* contains the message creation time value.

6.4 Authentication and authorization sample code

The `addUsernameToken` method demonstrates how to provide the security information to the SOAP header of the WSSE element. This method is common for the code scenarios provided in this document. It should be invoked before calling Web services in all scenarios.

```
/**  
  
 * A method to construct security information to the SOAP header of WSSE  
 element  
  
 * username should be in format of "UserName:GroupName"  
  
 */  
  
private void addUsernameToken(WSBindingProvider port, String username,  
String password) throws SOAPException {  
  
    String wsse_ns = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-  
wss-wssecurity-secext-1.0.xsd";  
  
    String wsu_ns = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-  
wss-wssecurity-utility-1.0.xsd";  
  
    SOAPFactory soapFactory = SOAPFactory.newInstance();  
  
    SOAPElement wsse = soapFactory.createElement("Security", "wsse",  
wsse_ns);  
  
    SOAPElement token = soapFactory.createElement("UsernameToken",
```

```
"wsse", wsse_ns);

    QName qname = new javax.xml.namespace.QName(wsu_ns, "Id", "wsu");

    // Set the wsu:Id to a unique value

    token.addAttribute(qname, "UsernameToken-6165421");

    SOAPElement usernameEl = soapFactory.createElement("Username",
"wsse", wsse_ns);

    // Set the username based on the UserName:GroupName that has been
assigned to the 3rd

    // party. This value will be assigned to the 3rd party by AT&T during the

    // Onboarding process.

    usernameEl.addTextNode(username);

    SOAPElement passwordEl = soapFactory.createElement("Password",
"wsse", wsse_ns);

    // Set the password based on the value that has been assigned to the 3rd
party

    passwordEl.addTextNode(password);

    token.addChildElement(usernameEl);

    token.addChildElement(passwordEl);

    wsse.addChildElement(token);

    port.setOutboundHeaders(Headers.create(wsse));

}
```

Refer to the *AT&T Network Services API Developer Guide* for further details.
Find it on the AT&T Developer Program website at <http://developer.att.com>.

7. Security

Access to AT&T production network services requires two-way SSL for an application server using the Network Service Application Pattern. The application must authenticate using a WSSE (user name and password) authentication token.

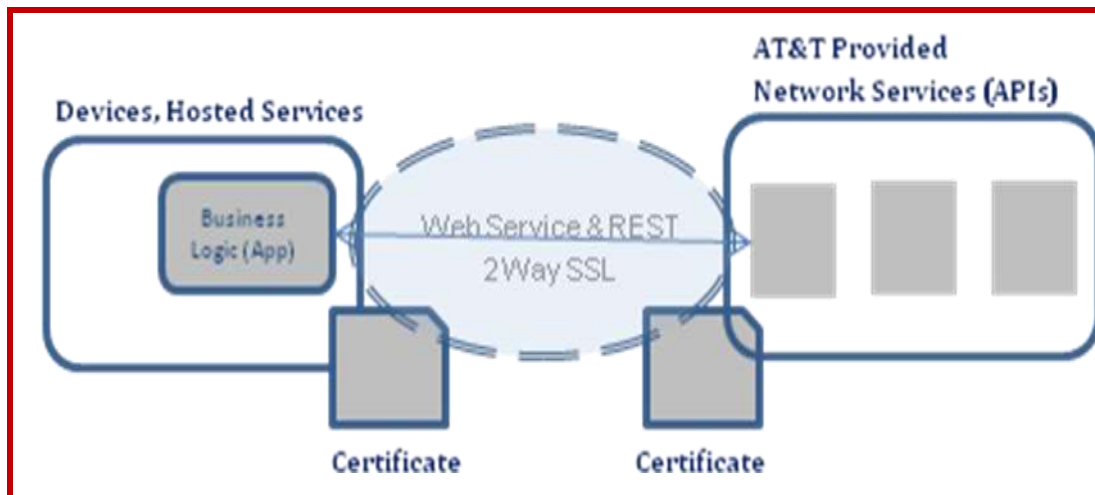


Figure 3. Successful Application Originated–Connection Established

7.1 HTML escape data persisted or sent through network services

We recommend that you guard against injection vulnerabilities to prevent the pass through of XML data that can executed directly.

7.2 Error auditing

We recommended that you audit error messages and faults to prevent exposing any compromising information that attackers (or end users) may use.

7.3 Explicitly control size of data being sent or received

We recommend that you set the maxOccurs attribute of an XML element to a suitably large value instead of unbounded to prevent denial of service attacks.

7.4 Protect against XML schema validate attacks

We recommend that developers do not set the processContents attribute of an XML element to “lax” or “skip” to prevent validation attacks.

7.5 Limit feature access

For security control when developing applications, we recommend that you limit access to each end user by using a role mechanism.

Consider a role-based scheme to make sure features are available only to only those that really need them. For example, if there is a LBS lookup feature, consider whether to give this only managers and not general customer service representatives. This prevents unintentional data access. It also saves network service requests and costs.

We encourage all application developers to read the network security guidelines section of the AT&T Developer Program website at <http://developer.att.com>.

8. Tools

8.1 Development tools

You may consider using tools from these third party vendors:

- Microsoft
- IBM
- Oracle, including Sun
- Open source foundations

You may consider using these languages:

- [Java](#)
- [Visual Basic.Net](#), [C++](#), [C#](#), and [Visual Basic 6.0](#)
- [ASP.Net](#)
- Scripting environments with HTTPS, XML, or SOAP such as [Perl](#), [Python](#), [Ruby](#)

You may consider using these development tools:

- [Eclipse](#) for development in [Java](#) or [Python](#)
- [Microsoft Visual Studio](#) for Visual Basic, C++, C#, and ASP.Net
- [Web Services Tools from Altova](#) (famous for XML Spy)
- [Stylus Studio](#)
- [Adobe Flex](#)
- [<oxygen/> xml editor](#)

8.2 Testing tools

Test applications from different points of view or perspectives. This helps to identify issues arising from different points of interaction with the Web services. For example:

- When acting as a service provider, server-side applications designed to broker network services transactions should be able to provide the results of an application-originated network services API request/response transaction when requested by an agent or client application.
- When acting as a service consumer, an application should be able to discover and invoke other Web services and use the functionality provided by the other Web services.
- Discover and invoke Web services through the service registry.

8.2.1 Quality management tool use

We recommend that you use an appropriate quality management tool to manage the test requirements, test cases and steps, requirement traceability matrix, automation scripts, test run results, and defects.

8.2.2 Test from all points of view

We recommend that you test server-side application logic implemented as a Web service from the points of view of all stakeholders. This includes roles such as a consumer, provider, and registrar.

8.2.3 Evaluate categories

For applications with network service awareness, we recommend that you evaluate the categories of testing outlined in this section, as applicable.

Fundamental test set:

- Individual component testing
- Service testing.
- Service publishing and discovery.
- WSDL and UDDI.
- Service communication (SOAP).
- Transport for service (protocols).
- HTTP and HPPS, JMS, Active-MQ.

Integration and end-to-end workflow testing (for server-side development):

- Service-to-service integration.
- Service-to-business process integration.
- Service-to-ESB integration.
- Service-to-low level infrastructure integration.
- Business process testing.

Positive and negative testing, and boundary value testing:

- Test SOAP messages and responses.

Security testing:

- Attaching a SOAP message, send modified message to service provider to get confidential information or perform a transaction.
- Modify the SOAP message while traveling between service consumer and provider.
- Attach SOAP message and read all confidential information.

WSDL scanning, parameter tampering:

- Test recursive payloads, replay attacks.
- Test interoperability.
- Ensure that the published Web service operations interoperate across various platforms.
- Verify platform and language independence.
- Enables reuse of existing systems.
- Test service on various platforms such as Windows, UNIX, and Linux.
- Test invoking services using various languages such as Java, C, and C++.

W3C, WS-I standards testing

Client's custom standards testing

Test for consistency, and atomicity

Performance testing:

- Latency, Transactional throughput, Utilization
- Size of SOAP/XML Message.
- Security Tokens inside SOAP/XML Message.
- SOAP with Attachments
- Tasks performed such as Validation, Transformation
- Number of clients sending SOAP/XML messages

Transaction testing

- Test Atomicity. In case of successful transactions, all operations are done. In case of failure, none of the operations has performed.

Test consistency

- Perform valid state transitions at completion.

Test isolation

- Verify the non-sharing of effects of the operations outside of the transaction until successful completion.

Service reliability

Security

Test for identification, integrity, authentication, and authorization.