



---

nCode Web Services Edition v4

# API Reference

Nova Marketing Group, Inc.

---

This document contains information proprietary to Nova Marketing Group Inc. and shall not be reproduced, copied, stored in a retrieval system, or transferred to other documents or media, or disclosed to others or used for any purpose other than that for which it is furnished, without the prior written permission of Nova Marketing Group Inc.

All reasonable efforts have been made to ensure that the information provided in this document is correct and up-to-date. However, Nova Marketing Group Inc disclaims all implied and/or express warranties and makes no representation as to the accuracy or completeness of any information in this document. Nova Marketing Group Inc. takes no responsibility and assumes no liability for the content of this document, including without limitation any mistake, error, omission, infringement, defamation, falsehood, or other material or omission that might offend or otherwise give rise to any claim or complaint.

© 1993-2025 NOVA MARKETING GROUP INC.

Nova Marketing Group Inc.  
4278 Greybrook Cres  
Mississauga, ON L4W 3G7  
Canada

(416) 878-0734

[www.novamg.com](http://www.novamg.com)

---

# Contents

<b>Introduction</b>	<b>1</b>
What is nCode?.....	1
What is nCode Web Services Edition? .....	2
Features.....	2
Using nCode .....	3
About this Document.....	4
About Nova Marketing Group.....	4
Copyright Notice .....	4
 <b>Working with nCode Web Services</b>	 <b>5</b>
System Architecture.....	5
nCode Web Services Edition Design Notes.....	5
Configuring nCode .....	6
nCode Settings Overview .....	7
Analysis Modes Explained .....	7
Working with Analysis Modes .....	9
Correction Styles Explained .....	9
Working with Correction Styles .....	11
Address Layouts Explained .....	11
Working with Address Layouts .....	12
 <b>nCode Web Services - Interfaces</b>	 <b>13</b>
nCodeWS4 Web Service .....	14
InCodeWS4 Web Service Interface .....	14
nCodeWS4R Web Service .....	15
InCodeWS4R Web Service Interface .....	15
nCodeWS4G Web Service.....	17
InCodeWS4G Web Service Interface .....	18
 <b>nCode Web Services - Source Code Samples</b>	 <b>19</b>
Source Code Samples .....	19

nCodeWS4CSClient Sample .....	20
nCodeWS4CSRestClient Sample .....	22
nCodeWS4JavaClient Sample .....	23
nCodeWS4JavaRestClient Sample .....	24
nCodeWS4RESTful Sample .....	25
Source Code Examples In C#.NET .....	26
Address Layout Details Example in C#.NET .....	26
Address Correction Example in C#.NET .....	29
Address Transformation Example in C#.NET .....	31
Address Browsing Example in C#.NET .....	33
Address Searching Example in C#.NET .....	43

## **nCodeWS4 Methods Reference 47**

AnalyzeAddress Method.....	48
AnalyzeGetAlternatives Method .....	49
AnalyzeRecords .....	50
TransformAddress Method .....	51
FormatAddress Method .....	51
FormatPE Method.....	52
SearchForAddresses Method .....	53
SearchForNames Method .....	54
SearchForAltMunicipalities Method .....	54
SearchForAltStreets Method.....	55
GetKeywords Method.....	56
ParseAddress Method .....	57
BrowseAddressEx Method .....	58
OneStopAnalyze Method.....	59
GetParamList Method.....	60
GetParamListEx Method .....	61
RemoveParameter Method .....	62
GetLayoutDetails Method.....	62
GetAnalysisMode Method .....	63
GetCorrectionStyle Method.....	63
SetAddressLayout Method .....	64
SetAnalysisMode Method.....	64
SetCorrectionStyle Method.....	65
GetWebServiceInfo Method .....	66

## nCodeWS4 Structures Reference 67

LineContent Structure.....	67
CStyleChange Structure.....	67
ParserElements Structure.....	68
AddressElements Structure.....	69
AnalyzeInput Structure.....	70
AnalyzeOutput Structure .....	70
AnalyzeGetAlternativesInput Structure.....	71
ALTMessage Structure.....	71
AlternativeAddress Structure.....	72
AnalyzeGetAlternativesOutput Structure .....	72
FormatAddressInput Structure.....	72
FormatAddressOutput Structure .....	73
FormatPEInput Structure .....	73
FormatPEOutput Structure .....	73
ParseAddressInput Structure.....	74
ParseAddressOutput Structure .....	74
BrowseAddressExInput Structure.....	74
BrowseAddressExOutput Structure.....	75
TransformInput Structure .....	75
TransformOutput Structure.....	75
BrowsingRequest Structure .....	76
NameElements Structure .....	77
AddressesFound Structure .....	78
NamesFound Structure .....	78
NAMAnalysisModeGen Structure.....	78
NAMAnalysisModeType Structure.....	79
AnalysisModeOutput Structure .....	79
AnalysisModeInput Structure .....	79
OperationResult Structure.....	80
CorrectionStyleOutput Structure .....	80
NCSCorrectionStyle Structure.....	80
KWDescription Structure .....	81
KWRequest Structure .....	81
GetKeywordsOutput Structure .....	82
AddressLine Structure .....	82
AddressLayout Structure .....	82

NewAddressLayout Structure .....	83
ParamList Structure .....	83
RemoveParamRequest Structure .....	83
GetParamRequest Structure .....	83
AltMncRecord Structure .....	84
AltMncSearchInput Structure .....	84
AltMncSearchOutput Structure .....	84
AltStrRecord Structure .....	84
AltStrSearchInput Structure .....	85
AltStrSearchOutput Structure .....	85
GuessingModes Structure .....	85
OneStopAnalyzeInput Structure .....	86
OneStopAnalyzeOutput Structure .....	87
WebServiceInfo Structure .....	88
AnalyzeRecordsInput .....	89
ResultingRecord .....	89
AnalyzeRecordsOutput .....	89

## **nCodeWS4 Enumerations Reference 91**

AES_LVRLanguage Enumeration .....	91
AES_SequenceCode Enumeration .....	91
AES_TypeOfAddress Enumeration .....	91
AES_TypeOfLVR Enumeration .....	92
AltListType Enumeration .....	92
AltMncSearchResult Enumeration .....	92
AltMncType Enumeration .....	93
AltMncValidityIndicator Enumeration .....	93
AltStrSearchResult Enumeration .....	93
ALT_MessageCategory Enumeration .....	94
ALT_MessageDetail Enumeration .....	94
CStyleCategory Enumeration .....	95
CStyleChangeType Enumeration .....	96
DAB_ResultType Enumeration .....	97
KW_CPCDBPresent Enumeration .....	97
KW_Language Enumeration .....	97
KW_OptimumOnly Enumeration .....	98
NAT_AddressType Enumeration .....	98

NAA_AnalysisType Enumeration .....	99
NAM_ChangedKeyElements Enumeration .....	99
NAM_MissingRouteInfoMode Enumeration .....	99
NB_AllInRange Enumeration .....	100
NB_IsIncluded Enumeration .....	100
NB_TargetType Enumeration .....	100
NAM_AcceptUnit enumeration .....	100
NAM_PropertyFlag Enumeration .....	100
NAM_SpellingLevel Enumeration .....	101
NAM_MissingWordsMode Enumeration .....	101
NAM_SOACompliant Enumeration .....	101
NCS_Lettercase Enumeration .....	101
NCS_Accent Enumeration .....	102
NCS_UnitForm Enumeration .....	102
NCS_Punctuation Enumeration .....	102
NCS_PCForm Enumeration .....	102
NCS_ExtraInfo Enumeration .....	102
NCS_KWOptimum Enumeration .....	103
NCS_MSOptimum Enumeration .....	103
NCS_PushLines Enumeration .....	103
NCS_StreetNameOptimum Enumeration .....	103
NCS_MunicipalityNameOptimum Enumeration .....	103
NCS_SOACompliant Enumeration .....	104
NPAR_ParameterType Enumeration .....	104
NAM_LanguageMode Enumeration .....	104

## **Appendix A – Address Status 107**

Address Status .....	107
Primary Address Status .....	107
Secondary Address Status .....	107
Urban Address Status .....	108

## **Appendix B – Address Lines 109**

Address Lines .....	109
Address Line Example .....	109

## **Glossary of Terms 111**





# Introduction

---

## What is nCode?

nCode is a fast and powerful tool for working with Canadian postal addresses.

You can use nCode to give your applications address validation, address correction, address database searching, address transformation and address capture capabilities. The nCode Programmers Toolkit consists of a set of functions (the nCode API) packaged into libraries (depending on your platform, DLLs, shared libraries, service programs) plus the nCode address reference database, built using data provided by Canada Post. The nCode API is available in versions that run natively on all major platforms, including Windows, Linux, Sun Solaris, HP-UX, AIX, AS/400 and MVS.

The nCode API provides functions which will ensure your addresses are complete, accurate, and in the optimal format for Canada Post. nCode is both flexible and powerful. You can take advantage of as many or as few of nCode's powerful features as you need to meet your address quality objectives.

nCode offers a number of advantages over other Canadian address quality software:

- **nCode is fast.** Our highly compressed address reference database is up to 10 times smaller than some other products. This means less physical I/O and better throughput. In fact, you can pre-load our entire 18 MB address reference database into RAM to completely avoid physical I/O while processing addresses.
- **nCode is flexible.** You work with your addresses the way you want to when you use nCode. nCode is architected from the ground up to be flexible and platform independent. There is no easier way to provide your own in-house applications with high-quality address management capabilities than to use nCode.
- **nCode is configurable.** No matter how many different systems you use addresses in and no matter how many address formats you need to use, nCode works with your data in your formats. No other Canadian address quality software is more configurable than nCode. With nCode you work with your addresses the way you want to.
- **nCode is easy to use.** You can use nCode with any platform and any programming tools. The nCode Programmers Toolkit is built using platform independent C++ and can be called by any programming tool that conforms to C language calling conventions. Whether you are using C/C++, Java, .NET, Perl, Ruby, COBOL or RPG, nCode is easy for you to integrate within your applications.
- **nCode is SERP certified by Canada Post,** so you know that you will be achieving the highest levels of address accuracy when you use nCode.

---

# What is nCode Web Services Edition?

nCode Web Services Edition, version 4, is an address quality toolset that allows your distributed applications to validate, correct, transform, capture and search for Canadian postal addresses.

nCode Web Services Edition is based on the powerful nCode Programmers Toolkit and provides all of the power and flexibility of the toolkit with an interface that is optimized for distributed, standards-based systems.

nCode Web Services Edition provides a number of interfaces:

- nCodeWS4 is a SOAP web service, which provides functional equivalent to the native nCode API. It is based on nCodeWS3 web service, which was the previous generation of nCode Web Services.
- nCodeWS4R is a RESTful web service, which provides most of the functionality that nCodeWS4 provides.
- nCodeWS4G is a RESTful Web Service which represents a REST/JSON HTTP GET equivalent to the nCodeWS4 Web Service. It is a subset of nCodeWS4R Web Service

nCode Web Services Edition runs on both Windows and Linux web and application servers. It can be accessed from any system on virtually any platform.

On the Windows side, nCode Web Services Edition was implemented using WCF (Windows Communication Foundation) as nCodeServer4. Users can use the full power and flexibility of WCF to customize how nCode Web Services Edition is exposed to users.

On the Linux side, nCode Web Services Edition implementation is equivalent to WCF implementation. It was implemented as nCodeServer4J, and it is based on Java, Tomcat, Jersey JAX-RS and Metro JAX-WS libraries.

---

## Features

nCode has many features that allow you to improve the accuracy of your addresses and the efficiency of your operations, saving your organization time and money while improving your customer service.

### Validation & Correction

nCode detects and automatically corrects problems with your address data. If you have an address which is too ambiguous to correct automatically, nCode lets you pick from a list of possible matches.

### Address Browsing

Use input address information to find valid addresses that match your criteria.

### Address Searching

Use any part or parts of an address to find valid addresses that match your criteria. You can ask nCode to list matches at various levels, including postal code ranges, streets, municipalities, forward sortation areas (FSAs) or provinces.

### Address Parsing

nCode can analyze your input address information and identify which words can be used in which address element contexts. For example, nCode can tell you that QUEBEC may be a province, a municipality or a street name.

### Address Transformation

nCode supports multiple simultaneous address layouts. nCode can quickly reformat any valid or correctable address into any Address Layout which you define. You have complete control over the formatting of addresses including forcing or suppressing punctuation, upper or mixed case letters, postal codes with or without a space, amongst many others.

### Address Capture

oneStop address capture combines address browsing and address analysis, enabling you to capture an address with as little typing as possible and in as few steps as possible.

## Technical Features

- **Reference database.** nCode uses sophisticated compression algorithms to bring its address reference database down to about 20 MB. This means unmatched performance in batch address correction and online address lookup applications. nCode's database technology is proprietary to Nova. There are no third-party database dependencies when using nCode. This means that your costs are kept low and your system management is kept simple.
- **Flexibility.** nCode has an open architecture, consisting of libraries which you can call from your own systems. This allows you to take maximum advantage of the features you need without forcing you to work with (or around) features which are not important to you.
- **Configurability.** nCode allows you to define your own address format characteristics as well as setting parameters which control nCode's address analysis and correction features. You can set these parameters in advance using nCode's configuration utility or you can set them dynamically from your application using nCode's configuration API functions.
- **Ease of use.** nCode comes with working sample programs. These samples show you how to use popular development languages, like C++, C# or Visual Basic, to interface with nCode in your environment. All of nCode's capabilities take advantage of a common database and set of DLLs. nCode Web Services Edition includes all of the WSDL you need to write reliable code against the SOAP API.
- **Rapid deployment.** nCode Web Services Edition provides you with all of the distributed system "plumbing" that you require to give your SOA systems top-notch address quality functionality quickly and easily. Our software is robust, scalable and proven by years of stable performance.

---

## Using nCode

nCode is useful in a wide variety of applications. Typical applications for nCode include:

- Call centers
- Customer relationship management
- Customer self-service websites
- Data capture
- Data warehousing
- External organization interfaces, e.g. credit bureaus
- Marketing
- Mailing

Anywhere you gather, manipulate or output addresses is a good place to use nCode. The greatest benefits can be realized by implementing nCode in your front-end data capture processes. This allows you to catch problems with address data before they become business problems for your organization. If this is not feasible, then nCode can easily help you to implement robust and powerful post-processing address quality routines.

### Getting Started with nCode Web Services Edition

Using nCode is simple. Any language that can access a SOAP/XML web service or a REST/JSON web service can access the nCode Web Services API. This includes popular development tools such as C# and VB.NET, Java, Ruby, Perl and many others. REST/JSON web services can also be accessed directly from JavaScript running in a Web Browser. The nCode Web Services API is stateless and scalable, so it can be used with confidence in environments with any number of users.

The nCode Web Services Edition installation installs an older version of the address database on the server where nCode will be running. Monthly refreshes of the address database are provided by Nova Marketing Group based on data from Canada Post, so your data will always be up to date.

Your application simply calls the nCode Web Service API web service methods that it requires based on your business needs. Various methods provide the ability to analyze and manipulate address information, while others provide control over how nCode performs this analysis and manipulation.

Detailed descriptions of the various nCode Web Service methods can be found later in this document.

---

## About this Document

This document is a complete function reference for the nCode Web Services Edition Canadian postal address quality toolkit.

The document includes information on the various functions within the nCode Web Services Edition API, including:

- Address Validation and Correction Methods
- Address Browsing and Parsing Methods
- Address Searching Methods
- Address Transformation Methods
- Parameter Methods
- Address Capture Methods
- Other Methods

This document was last modified on February 21, 2025.

---

## About Nova Marketing Group

Nova Marketing Group has been providing high quality software and systems consulting services since 1986. Nova Marketing Group is focused on address quality software solutions for the Canadian and U.S. marketplaces. In addition to nCode, Nova Marketing Group offers the nCode Programmers Toolkit and the nCode Batch Processor, as well as other products that help you maximize the value of your address database assets while minimizing the cost of address data management. You can learn more about Nova Marketing Group at [www.novamg.com](http://www.novamg.com).

---

## Copyright Notice

This document contains information proprietary to Nova Marketing Group Inc. and shall not be reproduced, copied, stored in a retrieval system, or transferred to other documents or media, or disclosed to others or used for any purpose other than that for which it is furnished, without the prior written permission of Nova Marketing Group Inc.

All reasonable efforts have been made to ensure that the information provided in this document is correct and up to date. However, Nova Marketing Group Inc. disclaims all implied and/or express warranties and makes no representation as to the accuracy or completeness of any information in this document. Nova Marketing Group Inc. takes no responsibility and assumes no liability for the content of this document, including without limitation any mistake, error, omission, infringement, defamation, falsehood, or other material omission that might offend or otherwise give rise to any claim or complaint.

© 1993 – 2025, NOVA MARKETING GROUP INC.

Postal Code<sup>OM</sup>, FSA<sup>OM</sup>, and LDU<sup>OM</sup> are official marks of Canada Post Corporation. Code postal<sup>MO</sup>, RTA<sup>MO</sup>, et UDL<sup>MO</sup> sont des marques officielles de la Société canadienne des postes.

# Working with nCode Web Services

---

## System Architecture

The logical system architecture of nCode Web Services Edition is as follows:

- The core of the system are nCode native libraries (DLLs or Shared Libraries), which use the nCode reference database. The nCode reference database is built using Nova's proprietary database technology and there are no third party dependencies, such as for database management systems.
- On the Windows side, WCF (Windows Communication Foundation) is used to expose both SOAP/XML and REST/JSON interfaces to the public. A third party component, Json.NET, is used for JSON formatting. The server nCodeServer4 is implemented as a Windows service, based on nCode native libraries. Both nCode and WCF behaviours can be further configured via application configuration file.
- On the Linux side, the server nCodeServer4J is implemented using Java, which calls nCode native libraries over their JNI wrappers. Metro JAX-WS library is used to expose SOAP/XML interfaces to the public. Jersey JAX-RS library is used to expose REST/JSON interfaces to the public. The server runs a number of servlets under Tomcat. nCode behaviours can be configured via nCode configuration file. The standard web.xml file is used to configure servlet behaviour.
- SOAP/XML clients use nCodeWS4 WSDL to generate client proxies in their programming language of choice. Server installation brings C# and Java sample projects and source code that demonstrate how to call nCodeWS4 Web service methods from these languages.
- REST/JSON clients are expected to be Web browser based and utilize Ajax, jQuery and JavaScript. Server installation brings a number of demo Web pages that demonstrate how to call nCodeWS4R RESTful web service methods in this context.

---

## nCode Web Services Edition Design Notes

The nCode Web Services API was designed with the following in mind:

- nCodeWS4 web service covers a wide range of functionality related to Canadian postal addressing, including address analysis, address browsing, address searching, address capture and address transformation.

- nCodeWS4R RESTful web service represents REST/JSON equivalent to nCodeWS4. The method signatures of nCodeWS4R are JSON equivalents to SOAP/XML based methods of nCodeWS4.
- nCodeWS4R RESTful web service implements cross-origin resource sharing (CORS), making cross domain calls from Web browsers possible.
- The methods of nCodeWS4 are independent; they can be called in any order with no impact on the correctness of the operation.
- Each method has only one parameter which represents the user's request, while the return value represents the web service method response.
- All the requests/responses translate directly into structures/classes in modern object-oriented languages like C#, VB.NET or Java. No additional XML parsing by the caller is required if a WSDL-generated proxy is used.
- The API is intended to be as simple as possible. In places where complexity could not be avoided, it is expressed through complex data structures rather than by increasing the number of methods.
- There is consistent use of only a few simple data types like strings and integers. The predefined choices are expressed as enumerations, while variable length data is given through arrays of either simple or complex data types.
- nCodeWS4 APIs are stateless implementations of nCode Programmers Toolkit APIs. In some cases, it might be helpful to understand what the underlying nCode engine is doing. For more detailed information about the nCode Toolkit API, see the nCode Programmers Toolkit 12.0 API Reference.

---

## Configuring nCode

In order to work with nCode, you must have an understanding of the following nCode configuration concepts that are used throughout the nCodeWS4 methods. The nCode Configuration Utility (nCodeCfg.exe) allows you to manipulate the various settings used by the nCode Programmers Toolkit, nCode Batch Processor and nCode Web Services Edition.

From your PC, you can manipulate nCode's configuration settings files, in particular:

- GMODLIST.DAT – The analysis mode and correction styles settings file
- GPARLIST.DAT – The address layout settings file

The files created on your computer using the nCode Configuration Utility are binary compatible with all platforms supported by nCode and can simply be copied from your computer to any computer running nCode, including nCode Web Services Edition, on any platform.

More:

[nCode Settings Overview](#)

[Analysis Modes Explained](#)

[Working with Analysis Modes](#)

[Correction Styles Explained](#)

[Working with Correction Styles](#)

[Address Layouts Explained](#)

[Working with Address Layouts](#)

---

## nCode Settings Overview

nCode has many settings which allow you to configure its operation to meet your specific address quality needs. These settings are grouped into four major categories. For each category, you can configure a collection of these settings and store them together with a name of your choosing. You can define as many of these settings presets as you need. When working with nCode, you simply recall, by name, the preset you want to use for each category. If you are working with the nCode Programmers Toolkit, each of the multiple API sessions which you can create can have its own combination of presets. This allows you to work with multiple groups of settings simultaneously.

The main preset groups, which are used in most operations with nCode, are:

- **Analysis Modes.** These are settings which configure the address parsing and matching process.
- **Correction Styles.** These are settings which configure the address standardization and presentation process.
- **Address Layouts.** These are settings which let nCode know how big your address record is, how many logical lines it is subdivided into, how long each of these address lines are, and what address elements can be expected to be found on each of these lines.

While it is possible to maintain all of these preset settings at runtime using web service requests in the nCodeWS4 API, most nCode users prefer to determine their configuration settings as a design time exercise. In this case, the easiest way to manipulate nCode settings is with the nCode Configuration Utility, which is a Windows Desktop application. Using the nCode Configuration Utility, you can easily create all of the nCode settings presets that you will need to work with nCode Web Services Edition. To install your presets for use by nCodeWS4, simply copy your settings files from your desktop system to the computer where the nCode Server is running. Settings files created by the nCode Configuration Utility can be used directly by systems running nCode on any supported platform, including Windows, Linux, any other Unix, and even AS/400 and MVS. For more information regarding the nCode Configuration Utility, see the nCode Configuration Utility Users Guide.

---

## Analysis Modes Explained

An analysis mode is a group of settings that define how an address will be analyzed by nCode. Analysis mode settings are stored in the GMODLIST.DAT file.

For more information on how to manipulate analysis mode settings, see the nCode Configuration Utility Users Guide.

The individual settings in an analysis mode are:

### Can be Changed

This setting applies at the address element level. Each input address element has a setting. Address elements that can be changed are considered by nCode during analysis. As nCode examines possible matches for your input address it will consider making address element corrections or substitutions. In general, the valid alternative address that requires the least number of address element changes will be considered the best match by nCode.

If you mark an address element as “can’t change” then this address element will be enforced as input. Use this setting with caution. If you have a strictly limited data set that allows you to impose this restriction, then it makes sense to do so. If there is any chance of an address element being missing, incorrect or misspelled in your input data, then using “can’t change” could prevent nCode from making a correction. The advantage of marking an address element “can’t change” is that it will prevent nCode from considering some alternatives and will therefore provide a more precise set of address alternatives.

### Key Address Elements

This setting applies at the address element level. Each input address element has a setting. Only those elements that can be changed can be marked as a key address element.

This setting is used in combination with the Maximum number of Key Address Elements (KAEs). nCode will constrain its searching to alternatives which can be found without changing more than the allowable maximum number of KAEs. Use this setting to indicate address elements in which you have greater than average confidence.

## Maximum KAEs

This setting is the maximum number of Key Address Elements which can be changed by nCode when considering address alternatives. Alternatives which require changing more KAEs than are allowed will be excluded from the search. A maximum KAEs value of 2 to 4 is recommended for most applications.

## Separated by Spaces

This setting applies at the address element level. Each input address element has a setting. If an address element is marked as separated by spaces, then this element must have one or more blanks delimiting it from other address elements. If an address element is not marked separated by spaces, then other delimiters are allowed.

## Spelling Level

This is a setting which governs nCode's application of spelling rules while analyzing an address to find possible matching alternatives. The setting is a number from 1 (least strict) to 5 (most strict). Note that performance can be affected when using lower values for this setting (e.g., 1, 2, or 3). Unless your addresses are frequently missing their postal codes and there is a reasonable proportion of typographical errors in your street name data, a higher spelling level value, such as 4 or 5, is generally recommended.

The setting is applied by nCode as follows: The spelling level of an address alternative is the number of address elements which are changed or added by nCode in order to match the input address to that possible valid alternative. If the spelling level of the address is greater than or equal to the value of this setting, then misspellings in the street and station name will be detected by nCode. This will generally result in a larger list of valid alternatives being considered and offered by nCode in its results.

Note that misspelling rules will be applied only to addresses which are missing their postal codes. If a postal code is provided, then street/station searching will be constrained by the postal code rather than by spelling rules, and the spelling level setting will be ignored for the purposes of parsing such addresses.

Note also that the level of an address can be retrieved using the nCode Programmers Toolkit with the `avc_get_level()` function of the Core API.

## Input is Accented

This is a true/false setting which tells nCode whether or not to recognize accented characters in the input address. If input is accented is false, any accented characters that actually exist in the input will be ignored, possibly causing the word containing the accented character to be considered incorrect.

For platforms and systems which do not accommodate accented characters, setting this to false will prevent invalid characters from being considered during analysis.

## Use Alternate Keywords

This is a true/false setting which tells nCode whether or not to recognize non-standard values for address keywords, such as street type and unit keyword.

## Penalize Missing Route Info

This is a true/false setting which tells nCode whether or not to penalize missing route information in rural streets served by route records. This setting is new to nCode 11 and defaults to pre nCode 11 behaviour, which is to penalize missing route information. The other setting, which is to ignore missing route information, is most useful in address capture situations.

## Guess Missing Words In Street Names

This setting is new to nCode 11 and defaults to pre nCode 11 behaviour, which is to never guess missing words in street/station names. Guessing missing words can be useful in situations when street names are missing a part of their name. For example, if '1800 The Collegeway' was simply given as '1800 Collegeway', and the postal code was also missing in the address, guessing missing words would be the only way to resolve it. This setting can be turned on, so that guessing takes place always, only when postal code is missing and municipality present, or only when municipality is present, regardless of postal code.



---

## Working with Analysis Modes

You can use the nCode Configuration Utility to maintain analysis modes. Analysis modes created with nCodeCfg.exe are stored in the GMODLIST.DAT file. Changes made to your analysis modes can be deployed to any system running nCode on any platform by simply copying the GMODLIST.DAT file from your Windows PC to the computer where you are running nCode.

Note that you can also maintain analysis modes at runtime from your applications using either the nCode Programmers Toolkit's Settings API (anmode32.dll) or the nCode Web Services Edition's parameter manipulation APIs.

For more information about analysis mode settings and how they impact the operation of nCode, see [Analysis Modes Explained](#).

You can create as many analysis modes as are required to meet your address quality needs.

---

## Correction Styles Explained

A correction style is a group of settings that define how an address will be formatted after matching by nCode. Correction style settings are stored in the GMODLIST.DAT file.

For more information on how to manipulate correction style settings, see the nCode Configuration Utility Users Guide.

The individual settings in a correction style are:

### Unit Number Format

The unit number format setting controls the placement and style of any unit number information in the optimized address. Choices include forcing the unit number to hyphenated format, e.g.: 201-33 BLOOR ST E, forcing the unit number to keyword format, e.g.: 33 BLOOR ST E UNIT 201, or leaving the unit information in the same format as it was input.

### Postal Code Format

The postal code format setting controls how the postal code is presented in the optimized address. Choices include forcing a space between the FSA and LDU, e.g.: L4W 3G7, forcing no space between the FSA and LDU, e.g.: L4W3G7, or leaving the spacing in the same format as it was input.

Note that if "leave as is" is selected and no postal code is provided, the default behaviour is to include a space between the FSA and the LDU.

### Accents in Output

The accents in output setting controls whether or not accented characters are included in the optimized address. Choices include forcing accents to be included, forcing accents to be suppressed, and leaving accented characters in place if they are included in the input.

Note that if "leave as is" is selected and there are no accented characters in the input information, then no accents will be added to the output. This will be so even if there are missing address elements which are added by nCode that call for accents to be used.

### Letter Case Handling

The letter case handling setting controls the letter case of the optimized address. Choices include forcing all upper case letters, forcing lower (mixed) case, in which the first letter of each word is capitalized, and all letters in the province code and postal code are capitalized, or "leave as is".

With the “leave as is” option, if the majority of recognized address elements are found to have initial capitals, then the address will be optimized with initial capitals. If there are more recognized keywords which are presented in all lowercase or all uppercase letters, then the address will be optimized with all uppercase letters.

### **Push Lines**

The push lines setting controls how blank address lines are handled during address optimization. When you have an address layout with multiple street lines that are of equal length, you can tell nCode to move address line data up as high as possible or down as low as possible. Moving lines up (Push Up) is the default behaviour and it is what nCode has done prior to version 9.0. Since version 9.0, Canada Post SERP testing requirements (for SERP testing only, not statement of accuracy) call for these lines to be moved down (Push Down) instead. **NOTE:** This setting will have no effect if the address lines in question are of unequal length.

### **Enable Optimizations**

The enable optimizations setting is a yes/no setting which tells nCode whether or not to perform address optimization when the current correction style is used.

### **Remove Punctuation**

The remove punctuation setting is a yes/no setting which tells nCode whether or not to strip out any punctuation when optimizing the address. Canada Post guidelines indicate that punctuation should be removed. Whether or not you follow this guideline in any particular application will depend on your business requirements.

### **Optimize Standard Extra Information**

The optimize standard extra information setting is a yes/no setting which tells nCode whether to optimize any standard extra information found in an input address. Standard extra information is not required for a mailing address to be deliverable to the delivery point, but it may be useful in routing an item once it reaches the delivery point. Examples of standard extra information are sub-unit keywords such as “BASEMENT”.

When this setting is true, nCode will optimize any standard extra information keywords which it finds. Otherwise nCode will leave the standard extra information as it finds it.

### **Keyword Optimization**

The keyword optimization setting applies to all address elements that are standardized keywords recognized by Canada Post. For each such address element, you can configure whether or not any keywords of that type will be optimized to reflect the Canada Post standard form.

When this setting is true, nCode will optimize keywords of the given type to use the Canada Post sanctioned spelling. Otherwise, these keywords will be left in the form in which they were input.

### **Misspellings**

The keyword misspellings setting applies to all address elements that are standardized keywords recognized by Canada Post. For each such address element, you can configure whether or not any keywords of that type will be optimized to reflect the Canada Post sanctioned spelling.

When this setting is true, nCode will optimize keywords of the given type to use the Canada Post sanctioned spelling. Otherwise, these keywords will use the spelling with which they were input.

### **Street Names**

This is a true/false setting which tells nCode whether or not to optimize street names. A typical situation is when the input address contains a valid alternate street name, which does not match CPC database value. The default behaviour is not to optimize street names, which is the SERP compliant way of handling addresses. When street names get optimized, optimization includes both street types and street directions.

## Municipality Names

This is a true/false setting which tells nCode whether or not to optimize municipality names. A typical situation is when the input address contains a valid alternate municipality name, which does not match CPC database value. The default behaviour is not to optimize municipality names, which is the SERP compliant way of handling addresses.

---

## Working with Correction Styles

You can use the nCode Configuration Utility to maintain correction styles. Correction styles created with nCodeCfg.exe are stored in the GMODLIST.DAT file. Changes made to your correction styles can be deployed to any system running nCode on any platform by simply copying the GMODLIST.DAT file from your Windows PC to the computer where you are running nCode.

Note that you can also maintain correction styles at runtime from your applications using either the nCode Programmers Toolkit's Settings API (anmode32.dll) or the nCode Web Services Edition's parameter manipulation APIs.

For more information about correction style settings and how they impact the operation of nCode, see [Correction Styles Explained](#). You can create as many correction styles as are required to meet your address quality needs.

---

## Address Layouts Explained

An address layout is a configuration setting which describes how address information is arranged physically for processing by nCode. Address layout settings are stored in the GPARLIST.DAT file.

For more information on how to manipulate address layouts, including important information about the rules for defining address layouts, see the nCode Configuration Utility Users Guide.

### Address Structure

Addresses are comprised of address elements, such as street name, municipality, and postal code. The list of address elements is determined largely by Canada Post. Some address elements are used as part of both address input and output. These are the standard address elements. Some address elements are used only for output. These are informational in nature and include such items as from and to street number range end points. In nCode there are also some special purpose address elements that are used as short-forms for collections of individual address elements. These special purpose address elements include "Address Line" which is a placeholder for all regular elements except municipality, province and postal code, and "Free Form" which is a placeholder for all address elements. The special purpose address elements are provided merely as a convenience for the purposes of simplifying the address layout definition function.

Each address element may appear on one or more address lines. Address lines are strings of a known length. The length of each line is completely configurable by you according to your address quality needs. In addition to setting the length and descriptive name of each address line, you also tell nCode which address elements to "scope" to each line. Scoping an address element to an address line has two purposes. First, it gives nCode a hint about where to look for each type of address element. Second, it gives nCode direction about where you want the address elements to go when an address is corrected or optimized.

Address lines are grouped together into an address layout. Each address line is a string of a known (configurable) length. You accumulate your address lines into a single string to make one consolidated address record. This address record is also of a known length which is definable by you. You define the starting point within the address layout record for each address line. Note that address lines cannot overlap, but they can have empty space before, after and between them if desired.

The maximum allowable total length for an address layout is 1,024 characters.

You can define as many different address layouts as you require and you can transform addresses from any one layout to any other layout, as long as the address is valid or correctable. You can choose to use very highly granular address layouts, if this is what you require, or alternatively, you can use non-granular, free-form layouts, or anything in between.

### Address Layout Tips

Here are some tips for defining address layouts.

- Use the nCode Configuration Utility to define address layouts if you are not totally confident with all of the requirements for address layouts. The Address Layout Properties Dialog in the nCode Configuration Utility gives you comprehensive feedback about the state of your address layout definition including detailed error messages. See the nCode Configuration Utility Users Guide for more information about the rules for address layouts.
- Remember to allow at least two characters at the end of your address layout for the carriage return / line feed that is required for nCode to process your addresses when they are presented in the form of a fixed record length file. This step is not necessary if the address layout will only be used with one address at a time directly with the nCode Programmers Toolkit API or nCode Web Services API.
- If your layout includes a line with the “address status” element scoped to it, the address status code will be returned directly with your address. This can be helpful if you are processing addresses in a file or if you want to avoid calling the address status retrieval function in the nCode Core API.
- Don’t mix free form layout lines in the same layout with lines containing other types of address elements. This can create ambiguous analysis situations and may result in non-intuitive address matches. Address layouts that contain free form lines can include lines with addressee and address status elements. Other combinations are possible but care must be taken.
- If your business requirements permit it, using two lines which could contain address line elements is a good practice. This is because there are a number of Canada Post rules which dictate splitting up address elements into different lines when possible. This affects Street Served by Route addresses and addresses containing extra information in particular.

---

## Working with Address Layouts

You can use the nCode Configuration Utility to maintain address layouts. Address layouts created with nCodeCfg.exe are stored in the GPARLIST.DAT file. Changes made to your address layouts can be deployed to any system running nCode on any platform simply by copying the GPARLIST.DAT file from your Windows PC to the computer where you are running nCode.

Note that you can also maintain address layouts at runtime from your application using either the nCode Programmers Toolkit Layout API (param32.dll) or the nCode Web Services Edition’s parameter manipulation APIs.

For more information about address layouts and how they impact the operation of nCode, see [Address Layouts Explained](#).

You can create as many address layouts as are required to meet your address quality needs.

# nCode Web Services - Interfaces

The third edition of nCode Web Services provided only the nCodeWS3 Web Service, which represented a functional equivalent to the native nCode 10.0 API. nCodeWS3 Web Service was a basic SOAP/XML based web service over http or https, offering no additional Web Service security options. The fourth edition of nCode Web Services is based on nCode 12.0 API and provides the following web services:

- [nCodeWS4](#) is a SOAP/XML web service which covers a wide range of functionality related to Canadian postal addressing, including address analysis, address browsing, address searching, address capture and address transformation. It extends the nCodeWS3 Web Service, while preserving its backwards compatibility with nCodeWS3, except for the namespace changes. The upgrade process for current nCodeWS3 users is simple: they only need to rebuild their client proxies using nCodeWS4 WSDL and no changes to the existing client source code would be required.
- [nCodeWS4R](#) is a RESTful web service which represents a REST/JSON HTTP POST equivalent to the nCodeWS4 web service. Some of the nCodeWS4 web service methods that were deemed not important in the REST/JSON context were excluded from the nCodeWS4R web service.
- [nCodeWS4G](#) is a RESTful web service which represents a REST/JSON HTTP GET equivalent to the nCodeWS4 web service. It is a subset of nCodeWS4R web service.

The method signatures of nCodeWS4R RESTful Web service are REST/JSON equivalents to SOAP/XML based methods of nCodeWS4 Web service. That is possible due to the shared internal implementation of SOAP/XML and REST/JSON web service methods.

nCodeWS4R RESTful Web services support CORS (cross-origin resource sharing), which enables modern Web browsers to make cross domain calls to RESTful web services from Web pages running under domains that are different than the domains running RESTful web services. The only REST verbs supported are POST, GET and OPTIONS (needed to implement CORS). The GET verb is not supported for those Web methods whose request parameters are complex structures.

All of the nCode Web Services implement certain Web Service interfaces, which are collections of Web Service method declarations. Using WCF terminology, Web Service interfaces are also known as Service Contracts. The names of the Web Service interfaces are used in the WSDLs, WCF configuration files, etc.

**Note.** With the nCode 12.0 release, the **BrowseAddress()** Web method was removed from nCodeWS4 Web service. It was deprecated long ago, when the **BrowseAddressEx()** Web method was introduced. The associated nCodeWS4 WSDL version is 4.4.

**Note.** With the nCode 12.3 release, the **AnalyzeRecords()** Web method was added to nCodeWS4 Web service.

---

## nCodeWS4 Web Service

The nCodeWS4 SOAP/XML Web Service reflects the functionality provided with the native nCode API. The native nCode API is very granular and the main reason for this design was to ensure nCode portability across a variety of platforms including Windows, Linux, Unix, MVS, etc. Thanks to such a design, native nCode API can be called from practically any programming language on virtually any platform. A downside to such a design is that the native nCode API is not stateless, meaning that its methods need to be called in a specific order to achieve a certain result. Since SOAP/XML Web services can understand much more complex structures, the nCodeWS4 Web Service was designed to be stateless. The nCodeWS4 Web Service methods were designed to correspond to a major subset of the native nCode API functionality, like address analysis, address parsing, address browsing, address searching, address capture, address, transformation, etc.

The nCodeWS4 SOAP/XML Web Service extends the nCodeWS3 Web Service. Since the nCodeWS4 Web Service is based on nCode 11.0 API, the new nCode 11.0 API features were also reflected in the nCodeWS4 Web Service, relative to the nCodeWS3 Web Service:

- oneStop address capture functionality was added as a new method
- alternate municipality search functionality was added as a new method
- alternate street search functionality was added as a new method
- guessing modes were added to the requests of address browsing and address parsing methods
- new flags were added to native search requests to explicitly specify letter case and accent style of the search results
- new analysis mode settings were reflected in the analysis mode related methods
- new correction style settings were reflected in the correction style related methods

When an nCodeWS4 Web Service request structure contains new members, extending the nCodeWS3 Web Service request structure, backwards compatibility is maintained by the nCodeWS4 Web Service assigning sensible default values to the new request structure members, if left undefined. This scenario would happen if nCodeWS3 Web Service client application simply upgraded to nCodeWS4 Web Service without making any source code changes. For better control, the client source code could be updated to set the values of the new request structure members explicitly.

When an nCodeWS4 Web Service response structure contains new members, extending the nCodeWS3 Web Service response structure, backwards compatibility is maintained by the existing client code simply ignoring the new response structure members. Of course, the existing client code could be changed to take advantage of the newly available response structure members.

nCodeWS4 Web Service implements the [InCodeWS4](#) Web Service interface.

---

## InCodeWS4 Web Service Interface

The following table lists **InCodeWS4** Web Service interface methods:

Method Name	Method Description
<a href="#">AnalyzeAddress</a>	Address Correction and Validation
<a href="#">AnalyzeGetAlternatives</a>	Address Correction With Address Alternatives Returned
<a href="#">AnalyzeRecords</a>	Address Correction and Validation in batch mode
<a href="#">OneStopAnalyze</a>	oneStop Address Capture
<a href="#">TransformAddress</a>	Address Transformation From One Layout to Another

<a href="#">FormatAddress</a>	Convert Address Elements Structure Into Address Record
<a href="#">FormatPE</a>	Convert Parser Elements Structure Into Address Record
<a href="#">ParseAddress</a>	Address Parsing
<a href="#">BrowseAddressEx</a>	Address Browsing With Full Functionality
<a href="#">SearchForAddresses</a>	Search For Addresses Providing a Set of Conditions
<a href="#">SearchForNames</a>	Search For Names Providing a Set of Conditions
<a href="#">SearchForAltMunicipalities</a>	Search For Alternate Municipalities
<a href="#">SearchForAltStreets</a>	Search For Alternate Streets
<a href="#">GetKeywords</a>	Search For Alternate Keywords
<a href="#">GetParamList</a>	Get the List of Analysis Modes, Correction Styles or Address Layouts
<a href="#">RemoveParameter</a>	Remove Analysis Mode, Correction Style or Address Layout
<a href="#">GetAnalysisMode</a>	Get Analysis Mode Details
<a href="#">GetCorrectionStyle</a>	Get Correction Style Details
<a href="#">GetLayoutDetails</a>	Get Address Layout Details
<a href="#">SetAddressLayout</a>	Create or Update an Address Layout
<a href="#">SetAnalysisMode</a>	Create or Update an Analysis Mode
<a href="#">SetCorrectionStyle</a>	Create or Update a Correction Style
<a href="#">GetWebServiceInfo</a>	Get Version Information About the Web Service
<a href="#">GetParamListEx</a>	Get the list of parameters that belong to a specific owner

---

## nCodeWS4R Web Service

The nCodeWS4R RESTful Web Service implements a subset of functionality provided by the [nCodeWS4](#) Web Service. All of its methods implement HTTP POST verb.

The signature of the nCodeWS4R Web Service methods is equivalent to the signature of the corresponding nCodeWS4 Web Service methods. The only difference between them is that the objects representing method request and response parameters get serialized and deserialized differently. The nCodeWS4R Web Service uses JSON for serialization, while the nCodeWS4 Web Service uses SOAP/XML.

nCodeWS4R Web Service implements the [InCodeWS4R](#) Web Service interface.

---

## InCodeWS4R Web Service Interface

The following table lists **InCodeWS4R** Web Service interface methods:

Method Name	Method Description
<a href="#"><u>AnalyzeAddress</u></a>	Address Correction and Validation
<a href="#"><u>AnalyzeGetAlternatives</u></a>	Address Correction With Address Alternatives Returned
<a href="#"><u>AnalyzeRecords</u></a>	Address Correction and Validation in batch mode
<a href="#"><u>OneStopAnalyze</u></a>	oneStop Address Capture
<a href="#"><u>TransformAddress</u></a>	Address Transformation From One Layout to Another
<a href="#"><u>FormatAddress</u></a>	Convert Address Elements Structure Into Address Record
<a href="#"><u>FormatPE</u></a>	Convert Parser Elements Structure Into Address Record
<a href="#"><u>ParseAddress</u></a>	Address Parsing
<a href="#"><u>BrowseAddressEx</u></a>	Address Browsing With Full Functionality
<a href="#"><u>SearchForAddresses</u></a>	Search For Addresses Providing a Set of Conditions
<a href="#"><u>SearchForNames</u></a>	Search For Names Providing a Set of Conditions
<a href="#"><u>SearchForAltMunicipalities</u></a>	Search For Alternate Municipalities
<a href="#"><u>SearchForAltStreets</u></a>	Search For Alternate Streets
<a href="#"><u>GetKeywords</u></a>	Search For Alternate Keywords
<a href="#"><u>GetParamList</u></a>	Get the List of Analysis Modes, Correction Styles or Address Layouts
<a href="#"><u>RemoveParameter</u></a>	Remove Analysis Mode, Correction Style or Address Layout
<a href="#"><u>GetAnalysisMode</u></a>	Get Analysis Mode Details
<a href="#"><u>GetCorrectionStyle</u></a>	Get Correction Style Details
<a href="#"><u>GetLayoutDetails</u></a>	Get Address Layout Details
<a href="#"><u>SetAddressLayout</u></a>	Create or Update an Address Layout
<a href="#"><u>SetCorrectionStyle</u></a>	Create or Update a Correction Style
<a href="#"><u>SetAnalysisMode</u></a>	Create or Update an Analysis Mode
<a href="#"><u>GetWebServiceInfo</u></a>	Get Version Information About the Web Service
<a href="#"><u>GetParamListEx</u></a>	Get the list of parameters that belong to a specific owner



---

## nCodeWS4G Web Service

The nCodeWS4G RESTful Web Service implements a subset of functionality provided by the nCodeWS4R Web Service. All of its methods implement HTTP GET verb. The JSON response from nCodeWS4G Web Service is fully equivalent to the JSON response from nCodeWS4R Web Service. However, the input to nCodeWS4G Web Service is provided via URL query parameters.

The URL-s for nCodeWS4G will keep the same form as the nCodeWS4R URL-s:

- For nCodeServer4, the URL will be `http://<host>/nCodeWS4/GET`, where the nCodeWS4R has the URL `http://<host>/nCodeWS4/REST`
- For nCodeServer4J, the URL will be `http://<host>:<port>/nCodeServer4J/nCodeWS4G`, where the nCodeWS4R has the URL `http://<host>:<port>/nCodeServer4J/nCodeWS4R`

Some of the advantages of using nCodeWS4G over nCodeWS4R include:

- Web browsers can be used to test nCode Web Services directly via URL bar
- Request elements that are not passed via query parameters will be assigned default values by nCode Servers

The nCodeWS4G Web Service implements strict checks on both the query parameters and the values passed. The query parameters must match by letter case and their values for enum, Boolean and Integer types must be in the format expected. If any mismatches are detected, errors will be returned.

When passing arrays of input address lines, we will rely on the fact that the address lines will have unique names, distinguishing them from other query parameter names.

Here are some examples of invoking nCodeWS4G Web service, in case of its nCodeServer4 implementation:

`http://localhost/nCodeWS4/GET/GetWebServiceInfo`

`http://localhost/nCodeWS4/GET/GetParamList?type=NPAP_CorrectionStyles`

`http://localhost/nCodeWS4/GET/GetLayoutDetails?name=SERP TEST`

`http://localhost/nCodeWS4/GET/GetAnalysisMode?name=SERP TEST`

`http://localhost/nCodeWS4/GET/GetCorrectionStyle?name=SERP TEST`

`http://localhost/nCodeWS4/GET/GetKeywords?LangInd=KW_LanguageAll&OptOnlyInd=KW_OptimumOnlyYes&AddressType=NAT_STRDRCTN`

`http://localhost/nCodeWS4/GET/AnalyzeAddress?strLayout=oneStop Address Layout&Address Line=125 allanhurst drive&City Province PC=toronto on&AnalysisType=ADDRESS_VALIDATION`

`http://localhost/nCodeWS4/GET/AnalyzeGetAlternatives?strLayout=oneStop Address Layout&Address Line=125 allanhurst drive&City Province PC=toronto on`

`http://localhost/nCodeWS4/GET/ParseAddress?strLayout=oneStop Address Layout&Address Line=125 allanhurst drive&City Province PC=toronto on`

`http://localhost/nCodeWS4/GET/BrowseAddressEx?strLayout=oneStop Address Layout&Address Line=125 allanhurst drive&City Province PC=toronto on`

`http://localhost/nCodeWS4/GET/TransformAddress?strFirstLayout=oneStop Address Layout&strSecondLayout=SERP TEST&Address Line=125 allanhurst drive&City Province PC=toronto on`

`http://localhost/nCodeWS4/GET/OneStopAnalyze?AddressLayout=oneStop Address Layout&Address Line=125 allanhurst drive&City Province PC=toronto on`

`http://localhost/nCodeWS4/GET/SearchForNames?MaxItems=10&Name=DUNDAS&TargetType=NB_TargetNames`

`http://localhost/nCodeWS4/GET/SearchForAddresses?MaxItems=10&Name=DUNDAS`

nCodeWS4G Web Service implements the [InCodeWS4G](#) Web Service interface.

---

# InCodeWS4G Web Service Interface

The following table lists InCodeWS4G Web Service interface methods:

Method Name	Method Description
<a href="#">AnalyzeAddress</a>	Address Correction and Validation
<a href="#">AnalyzeGetAlternatives</a>	Address Correction With Address Alternatives Returned
<a href="#">OneStopAnalyze</a>	oneStop Address Capture
<a href="#">TransformAddress</a>	Address Transformation From One Layout to Another
<a href="#">ParseAddress</a>	Address Parsing
<a href="#">BrowseAddressEx</a>	Address Browsing With Full Functionality
<a href="#">SearchForAddresses</a>	Search For Addresses Providing a Set of Conditions
<a href="#">SearchForNames</a>	Search For Names Providing a Set of Conditions
<a href="#">SearchForAltMunicipalities</a>	Search For Alternate Municipalities
<a href="#">SearchForAltStreets</a>	Search For Alternate Streets
<a href="#">GetKeywords</a>	Search For Alternate Keywords
<a href="#">GetParamList</a>	Get the List of Analysis Modes, Correction Styles or Address Layouts
<a href="#">GetAnalysisMode</a>	Get Analysis Mode Details
<a href="#">GetCorrectionStyle</a>	Get Correction Style Details
<a href="#">GetLayoutDetails</a>	Get Address Layout Details
<a href="#">GetWebServiceInfo</a>	Get Version Information About the Web Service
<a href="#">GetParamListEx</a>	Get the list of parameters that belong to a specific owner

Note that some of the nCodeWS4 methods were omitted from the InCodeWS4G interface: FormatAddress(), FormatPE(), SetAddressLayout(), SetAnalysisMode(), SetCorrectionStyle(), RemoveParameter().

The reasons for the omission:

- These Web methods take complex input, like arrays of structures or structures with too many elements (AddressElements)
- The Set\*() methods are very rarely used, only when new address layouts, analysis modes and correction styles need to be created

The missing methods can be called using the InCodeWS4R interface, still staying within the RESTful framework. The URL size limit of 2K does not affect usage of InCodeWS4G, since even the most complex requests will have less than 30 fields \* 30 bytes query parameters length.

# nCode Web Services - Source Code Samples

---

## Source Code Samples

The nCodeServer4 installation brings with it a number of sample projects in various programming languages that demonstrate how to access and how to use the functionality provided by nCode Web Services:

- [nCodeWS4CSClient](#) is a C#.NET project which was developed with Visual Studio 2010. It demonstrates most of the nCodeWS4 Web Service functionality using SOAP, with the exclusion of creating address layouts, analysis modes and correction styles.
- [nCodeWS4CSRestClient](#) is a C#.NET project which was developed with Visual Studio 2010. It demonstrates most of the nCodeWS4 Web Service functionality using REST and JSON, with the exclusion of creating address layouts, analysis modes and correction styles.
- [nCodeWS4JavaClient](#) is a Java project which was developed with Eclipse Java EE IDE for Web Developers, targeting JavaSE-1.6(jre6). This project is a functional Java equivalent to nCodeWS4CSClient project.
- [nCodeWS4JavaRestClient](#) is a Java project which was developed with Eclipse Java EE IDE for Web Developers, targeting JavaSE-1.6(jre6). This project is a functional Java equivalent to nCodeWS4CSRestClient project.
- [nCodeWS4RESTful](#) project is a collection of sample HTML pages that combine JavaScript, AJAX and jQuery to demonstrate how nCodeWS4R Web Service functionality can be accessed directly from Web pages.

Please note the following:

- The nCodeServer4 installation brings with it nCodeWS4.wsdl, which can be found under the installation path ([Program Files]\Nova Marketing Group\nCodeServer4\WSDL). Alternatively, you can access the WSDL via the following URLs: <http://localhost/nCodeWS4Docs/nCodeWS4.wsdl>. Based on the WSDL, nCode Web Service proxy source code can be generated either directly from the development environments or by running dedicated tools, like wsdl.exe (.NET) or wsimport (Java).
- The nCodeWS4CSClient and nCodeWS4JavaClient projects contain nCodeWS4 proxy source code in their respective programming languages. When developing your own applications that access nCodeWS4 Web Service, you can either reuse the proxy code from these projects or generate your own nCodeWS4 proxy source code from nCodeWS4 WSDL.
- The proxy source code used in nCodeWS4CSClient and nCodeWS4JavaClient projects relies on the localhost domain name. As such, the sample projects are meant to be run on the machines where nCodeServer4 is installed. The sample projects could be modified to enable remote access to nCodeWS4 Web Service by changing localhost to the appropriate domain name in your environment.

- If you want to test nCodeWS4 Web Service methods directly, and you have installed Visual Studio, you can use wctestclient.exe application by passing it nCode Web Service URL <http://localhost/nCodeWS4>. The WCF Test Client application will then list all of the available methods in the left pane. Double clicking a method will enable you to enter the request data, execute the method and view the response. A formatted SOAP request and response can be viewed under the XML tab. Many other third party applications can be used for the same purpose.
- nCodeWS4RESTful project runs under IIS and was configured to access nCodeWS4R Web Service locally by using the localhost domain name. By editing nCodeWS4Urls.js under [IISROOT]\nCodeWS4RESTful folder, you can change the following definition:

```
var nCodeWS4BaseUrl = "http://localhost/nCodeWS4/REST/";
```

with the appropriate domain name in your environment. You can then run the pages remotely, using the following URL: <http://<domainname>/nCodeWS4RESTful/>.

These changes can also be accomplished by running Host Name Configuration application (as Admin) on the nCodeServer4 host machine. Please note that the nCodeServer4 installation optionally asks users if they want to automatically replace the default localhost with the machine name and if Yes is selected, the changes to the corresponding files (including nCodeWS4Urls.js) are automatically performed during the installation.

- The sample source code projects utilize the default nCodeServer4 installation, which runs nCode Web Services based on http, without additional Web Service security provided. However, the client side source code needed to access secure nCode Web Services stays the same, thanks to the flexibility of WCF configuration. The nCodeServer4 installation brings with it a number of sample nCodeServer4 configuration files (under [Program Files]\Nova Marketing Group\nCodeServer4\Sample Config), which demonstrate various WCF security scenarios, like Windows security, message security using user ID and password, certificates, etc. If you want to experiment with securing nCode Web Services using WCF configuration, you need to be familiar with WCF security and WCF configuration concepts. We recommend that you always back up the nCodeServer4 application configuration file ([Program Files]\Nova Marketing Group\nCodeServer4\bin\nCodeServer4Host.exe.config), before making any changes to it. If you do make changes to the nCodeServer4 application configuration file, you will need to restart the 'nCodeServer4 WCF Service Host' Windows Service. Once the nCodeServer4 configuration has been changed to run secure nCode Web Services, similar WCF client side configuration changes need to take place.
- A pre-condition for successfully running the samples is that the 'nCodeServer4 WCF Service Host' Windows Service is running. The nCodeServer4 installation starts the service as the last installation step. The service is configured with the startup type 'Automatic', meaning that it will start automatically in case of a system reboot. If in doubt, please use Windows management tools to verify the status of the service.

For each sample project, a table that links nCode Web Service methods with the source code files and the function names, which demonstrate the usage of the methods, is provided.

---

## nCodeWS4CSClient Sample

The nCodeWS4CSClient sample project was developed with Visual Studio 2010, targeting .NET Framework 4. It demonstrates how to access nCodeWS4 Web Service functionality using C#.NET. The nCodeServer4 installation brings with it a copy of the project as nCodeWS4CSClient.zip and installs it under [Program Files]\Nova Marketing Group\nCodeServer4\Sample Code. The sample source code has already been compiled into an executable, so that it can be run without having Visual Studio 2010 installed. There is still a dependency on .NET Framework 4. The related .NET executable nCodeWS4SampleClient.exe is packaged under the same installation folder. The application has a command line format and presents a menu with numerous options, like Address Analysis, Address Transformation, etc.

The following table links nCodeWS4 Web Service method names with the source code files and the function names, which demonstrate the usage of the Web Service methods:

Method Name	Source Code File	Function Name
-------------	------------------	---------------

<a href="#">AnalyzeAddress</a>	N/A	N/A
<a href="#">AnalyzeGetAlternatives</a>	nCodeWS4ClientAddress.cs	AnalyzeAddress
<a href="#">OneStopAnalyze</a>	nCodeWS4ClientAddress.cs	OneStopAnalyze
<a href="#">TransformAddress</a>	nCodeWS4ClientAddress.cs	TransformAddress
<a href="#">FormatAddress</a>	nCodeWS4ClientAddress.cs	FormatAddress
<a href="#">FormatPE</a>	N/A	N/A
<a href="#">ParseAddress</a>	nCodeWS4ClientAddress.cs	ParseAddress
<a href="#">BrowseAddressEx</a>	nCodeWS4ClientAddress.cs	BrowseAddress
<a href="#">SearchForAddresses</a>	nCodeWS4ClientSearches.cs	AddressDbSearch
<a href="#">SearchForNames</a>	nCodeWS4ClientSearches.cs	AddressDbSearch
<a href="#">SearchForAltMunicipalities</a>	nCodeWS4ClientSearches.cs	AltMunicipalitySearch
<a href="#">SearchForAltStreets</a>	nCodeWS4ClientSearches.cs	AltStreetSearch
<a href="#">GetKeywords</a>	nCodeWS4ClientSearches.cs	KeywordSearch
<a href="#">GetParamList</a>	nCodeWS4ClientParams.cs	PresentAddressLayouts
<a href="#">RemoveParameter</a>	N/A	N/A
<a href="#">GetAnalysisMode</a>	nCodeWS4ClientParams.cs	PresentAnalysisModes
<a href="#">GetCorrectionStyle</a>	nCodeWS4ClientParams.cs	PresentCorrectionStyles
<a href="#">GetLayoutDetails</a>	nCodeWS4ClientParams.cs	PresentAddressLayouts
<a href="#">SetAddressLayout</a>	N/A	N/A
<a href="#">SetAnalysisMode</a>	N/A	N/A
<a href="#">SetCorrectionStyle</a>	N/A	N/A
<a href="#">GetWebServiceInfo</a>	nCodeWS4ClientSearches.cs	GetWebServiceInfo

The reasons that some of the nCodeWS4 methods were not used in this sample project are the following:

- The [AnalyzeAddress](#) method is used the same way as the [AnalyzeGetAlternatives](#) method, except that the AnalyzeGetAlternatives method receives more information.
- The [FormatPE](#) method is better suited for a GUI based application; it is used in nCodeWS4Samples project instead.
- The [RemoveParameter](#) method is rarely used and its usage should be limited to managing lists of address layouts, analysis modes and correction styles.
- The [SetAddressLayout](#), [SetAnalysisMode](#) and [SetCorrectionStyle](#) methods require a rather sophisticated GUI; they are used in nCodeWS4Samples project instead.

The samples were written in a flexible manner, so that the client can dynamically point to a remote server implementing nCodeWS4 SOAP interface.

---

## nCodeWS4CSRestClient Sample

The nCodeWS4CSRestClient sample project was developed with Visual Studio 2010, targeting .NET Framework 4. It demonstrates how to access nCodeWS4 RESTful Web Service functionality using C#.NET. The nCodeServer4 installation brings with it a copy of the project as nCodeWS4CSRestClient.zip and installs it under [Program Files]\Nova Marketing Group\nCodeServer4\Sample Code. The application has a command line format and presents a menu with numerous options, like Address Analysis, Address Transformation, etc.

The following table links nCodeWS4 RESTful Web Service method names with the source code files and the function names, which demonstrate the usage of the Web Service methods:

Method Name	Source Code File	Function Name
<a href="#">AnalyzeAddress</a>	N/A	N/A
<a href="#">AnalyzeGetAlternatives</a>	nCodeWS4ClientAddress.cs	AnalyzeAddress
<a href="#">OneStopAnalyze</a>	nCodeWS4ClientAddress.cs	OneStopAnalyze
<a href="#">TransformAddress</a>	nCodeWS4ClientAddress.cs	TransformAddress
<a href="#">FormatAddress</a>	nCodeWS4ClientAddress.cs	FormatAddress
<a href="#">ParseAddress</a>	nCodeWS4ClientAddress.cs	ParseAddress
<a href="#">BrowseAddressEx</a>	nCodeWS4ClientAddress.cs	BrowseAddress
<a href="#">SearchForAddresses</a>	nCodeWS4ClientSearches.cs	AddressDbSearch
<a href="#">SearchForNames</a>	nCodeWS4ClientSearches.cs	AddressDbSearch
<a href="#">SearchForAltMunicipalities</a>	nCodeWS4ClientSearches.cs	AltMunicipalitySearch
<a href="#">SearchForAltStreets</a>	nCodeWS4ClientSearches.cs	AltStreetSearch
<a href="#">GetKeywords</a>	nCodeWS4ClientSearches.cs	KeywordSearch
<a href="#">GetParamList</a>	nCodeWS4ClientParams.cs	PresentAddressLayouts
<a href="#">GetAnalysisMode</a>	nCodeWS4ClientParams.cs	PresentAnalysisModes
<a href="#">GetCorrectionStyle</a>	nCodeWS4ClientParams.cs	PresentCorrectionStyles
<a href="#">GetLayoutDetails</a>	nCodeWS4ClientParams.cs	PresentAddressLayouts
<a href="#">GetWebServiceInfo</a>	nCodeWS4ClientSearches.cs	GetWebServiceInfo

The reasons that some of the nCodeWS4 RESTful methods were not used in this sample project are the following:

- The [AnalyzeAddress](#) method is used the same way as the [AnalyzeGetAlternatives](#) method, except that the AnalyzeGetAlternatives method receives more information.

The samples were written in a flexible manner, so that the client can dynamically point to a remote server implementing nCodeWS4 RESTful interface.

---

## nCodeWS4JavaClient Sample

The nCodeWS4JavaClient sample project was developed with Eclipse Java EE IDE for Web Developers, targeting JavaSE-1.6(jre6). It demonstrates how to access nCodeWS4 Web Service functionality using Java. The nCodeServer4 installation brings with it a copy of the project as nCodeWS4JavaClient.zip and installs it under [Program Files]\Nova Marketing Group\nCodeServer4\Sample Code. The sample source code has already been compiled so that it can be run without having Eclipse IDE installed. The Java executables were packaged in nCodeWS4JavaClient.jar under the same installation folder and can be run as java -jar nCodeWS4JavaClient.jar. The application has a command line format and presents a menu with numerous options, like Address Analysis, Address Transformation, etc.

The following table links nCodeWS4 Web Service method names with the source code files and the function names, which demonstrate the usage of the Web Service methods:

Method Name	Source Code File	Function Name
<a href="#">AnalyzeAddress</a>	N/A	N/A
<a href="#">AnalyzeGetAlternatives</a>	NCWS4ClientAnalyze.java	AnalyzeAddress
<a href="#">OneStopAnalyze</a>	NCWS4ClientOneStop.java	OneStopAnalysis
<a href="#">TransformAddress</a>	NCWS4ClientTransform.java	TransformAddress
<a href="#">FormatAddress</a>	NCWS4ClientFormat.java	FormatAddress
<a href="#">FormatPE</a>	N/A	N/A
<a href="#">ParseAddress</a>	NCWS4ClientParse.java	ParseAddress
<a href="#">BrowseAddressEx</a>	NCWS4ClientBrowse.java	BrowseAddress
<a href="#">SearchForAddresses</a>	NCWS4ClientSearch.java	AddressDbSearch
<a href="#">SearchForNames</a>	NCWS4ClientSearch.java	AddressDbSearch
<a href="#">SearchForAltMunicipalities</a>	NCWS4ClientAltMnc.java	AltMunicipalitySearch
<a href="#">SearchForAltStreets</a>	NCWS4ClientAltStr.java	AltStreetSearch
<a href="#">GetKeywords</a>	NCWS4ClientKeywords.java	KeywordSearch
<a href="#">GetParamList</a>	NCWS4ClientLayouts.java	PresentAddressLayouts
<a href="#">RemoveParameter</a>	N/A	N/A
<a href="#">GetAnalysisMode</a>	NCWS4ClientAModes.java	PresentAnalysisModes
<a href="#">GetCorrectionStyle</a>	NCWS4ClientCStyles.java	PresentCorrectionStyles
<a href="#">GetLayoutDetails</a>	NCWS4ClientLayouts.java	PresentAddressLayouts
<a href="#">SetAddressLayout</a>	N/A	N/A
<a href="#">SetAnalysisMode</a>	N/A	N/A
<a href="#">SetCorrectionStyle</a>	N/A	N/A



The reasons that some of the nCodeWS4 methods were not used in this sample project are the following:

- The [AnalyzeAddress](#) method is used the same way as the [AnalyzeGetAlternatives](#) method, except that the AnalyzeGetAlternatives method receives more information.
- The [FormatPE](#) method is better suited for a GUI based application; it is used in the nCodeWS4Samples project instead.
- The [RemoveParameter](#) method is rarely used and its usage should be limited to managing lists of address layouts, analysis modes and correction styles.
- The [SetAddressLayout](#), [SetAnalysisMode](#) and [SetCorrectionStyle](#) methods require a rather sophisticated GUI; they are used in nCodeWS4Samples project instead.

The samples were written in a flexible manner, so that the client can dynamically point to a remote server implementing nCodeWS4 SOAP interface.

Running the Java client sample remotely may also require host name change in the WSDL port definition. This can be accomplished by running Host Name Configuration application (as Admin) on the nCodeServer4 host machine. Please note that the nCodeServer4 installation optionally asks users if they want to automatically replace the default localhost with the machine name and if Yes is selected, the changes to the corresponding files (including WSDL-s) are automatically performed during the installation.

---

## nCodeWS4JavaRestClient Sample

The nCodeWS4JavaRestClient sample project was developed with Eclipse Java EE IDE for Web Developers, targeting JavaSE-1.6(jre6). It demonstrates how to access nCodeWS4 RESTful Web Service functionality using Java. The nCodeServer4 installation brings with it a copy of the project as nCodeWS4JavaRestClient.zip and installs it under [Program Files]\Nova Marketing Group\nCodeServer4\Sample Code. The application has a command line format and presents a menu with numerous options, like Address Analysis, Address Transformation, etc.

The following table links nCodeWS4 RESTful Web Service method names with the source code files and the function names, which demonstrate the usage of the Web Service methods:

Method Name	Source Code File	Function Name
<a href="#">AnalyzeAddress</a>	N/A	N/A
<a href="#">AnalyzeGetAlternatives</a>	NCWS4ClientAnalyze.java	AnalyzeAddress
<a href="#">OneStopAnalyze</a>	NCWS4ClientOneStop.java	OneStopAnalysis
<a href="#">TransformAddress</a>	NCWS4ClientTransform.java	TransformAddress
<a href="#">FormatAddress</a>	NCWS4ClientFormat.java	FormatAddress
<a href="#">ParseAddress</a>	NCWS4ClientParse.java	ParseAddress
<a href="#">BrowseAddressEx</a>	NCWS4ClientBrowse.java	BrowseAddress
<a href="#">SearchForAddresses</a>	NCWS4ClientSearch.java	AddressDbSearch
<a href="#">SearchForNames</a>	NCWS4ClientSearch.java	AddressDbSearch
<a href="#">SearchForAltMunicipalities</a>	NCWS4ClientAltMnc.java	AltMunicipalitySearch



<a href="#">SearchForAltStreets</a>	NCWS4ClientAltStr.java	AltStreetSearch
<a href="#">GetKeywords</a>	NCWS4ClientKeywords.java	KeywordSearch
<a href="#">GetParamList</a>	NCWS4ClientLayouts.java	PresentAddressLayouts
<a href="#">GetAnalysisMode</a>	NCWS4ClientAModes.java	PresentAnalysisModes
<a href="#">GetCorrectionStyle</a>	NCWS4ClientCStyles.java	PresentCorrectionStyles
<a href="#">GetLayoutDetails</a>	NCWS4ClientLayouts.java	PresentAddressLayouts
<a href="#">GetWebServiceInfo</a>	NCWS4ClientWSInfo.java	GetWebServiceInfo

The reasons that some of the nCodeWS4 RESTful methods were not used in this sample project are the following:

- The [AnalyzeAddress](#) method is used the same way as the [AnalyzeGetAlternatives](#) method, except that the AnalyzeGetAlternatives method receives more information.

The samples were written in a flexible manner, so that the client can dynamically point to a remote server implementing nCodeWS4 RESTful interface.

---

## nCodeWS4RESTful Sample

The nCodeWS4RESTful project was developed as a collection of sample HTML pages that combine JavaScript, AJAX and jQuery to demonstrate how nCodeWS4R Web Service functionality can be accessed directly from Web pages. The sample project installs under [IISROOT]\nCodeWS4RESTful. You can run the pages directly using the URL <http://localhost/nCodeWS4RESTful/>.

The following table links nCodeWS4R Web Service method names with the source code files and the function names, which demonstrate the usage of the Web Service methods:

Method Name	Source Code File	Function Name
<a href="#">AnalyzeAddress</a>	nCodeAnalysis.html	AnalyzeAddress
<a href="#">AnalyzeGetAlternatives</a>	nCodeAnalysis.html	AnalyzeGetAlternatives
<a href="#">OneStopAnalyze</a>	nCodeOneStop.html	ExecuteAddressLookup
<a href="#">TransformAddress</a>	nCodeTransformation.html	TransformAddress
<a href="#">ParseAddress</a>	nCodeBrowsing.html	ParseAddress
<a href="#">BrowseAddressEx</a>	nCodeBrowsing.html	BrowseAddress
<a href="#">SearchForAddresses</a>	nCodeSearching.html	SearchForAddresses
<a href="#">SearchForNames</a>	nCodeSearching.html	SearchForNames
<a href="#">GetKeywords</a>	nCodeKeywords.html	KWLookup
<a href="#">GetParamList</a>	nCodeParameters.html	ParametersLookup
<a href="#">GetAnalysisMode</a>	nCodeParameters.html	AnalysisModeDetails

<a href="#">GetCorrectionStyle</a>	nCodeParameters.html	CorrectionStyleDetails
<a href="#">GetLayoutDetails</a>	nCodeParameters.html	AddressLayoutDetails
<a href="#">GetWebServiceInfo</a>	nCodeWSInfo.html	GetWSInfo

---

## Source Code Examples In C#.NET

A number of source code examples in C#.NET are provided here, in order to show how easy it is to use nCode Web Services. All of the examples were developed and tested with Visual Studio.NET 2010, targeting .NET Framework 4 Client Profile. Before you can use the source code examples in your programs, you need to do the following:

- Add the service references to your project. For nCodeWS4 Web Service, specify the address as <http://localhost/nCodeWS4> and the namespace as nCodeWebService4. Adding the service references will automatically generate the Web Service proxy classes, which will be used to call nCode Web Service methods. The name of the proxy class will be InCodeWS4Client.
- Add the namespaces to the source code files that will be calling nCode Web Services. If your project name is nCodeWS4CSSamples, add the following namespace:

```
using nCodeWS4CSSamples.nCodeWebService4;
```

- Tweak the app.config file in your project, if necessary. When you add the service references to your project, Visual Studio will add the default WCF Client configuration that corresponds to the Web Service proxy classes generated. This will include the endpoints and the bindings they reference. However, the default binding values may sometimes not be sufficient. For example, if the size of the response exceeds 64K, WCF may throw the following exception:

System.ServiceModel.CommunicationException: The maximum message size quota for incoming messages (65536) has been exceeded. To increase the quota, use the MaxReceivedMessageSize property on the appropriate binding element.

You can resolve this by editing the app.config file and setting the maxReceivedMessageSize = "4194304", or any other number that is sufficient, for the basicHttpBinding configuration and the binding named BasicHttpBinding\_InCodeWS4.

The following source code examples in C#.NET are available:

[GetAddressLayoutDetailsTest](#) shows how to query address layout properties.

[AddressCorrectionTest](#) shows how to perform address correction.

[AddressTransformationTest](#) shows how to perform address transformation.

[AddressBrowsingTest](#) shows how to perform address browsing.

[AddressSearchTest](#) shows how to perform address searching.

---

## Address Layout Details Example in C#.NET

The GetAddressLayoutDetailsTest() example shows how to query address layout properties:

```
static void GetAddressLayoutDetailsTest ()
```

```

{

    InCodeWS4Client ws4 = null; // nCodeWS4 proxy reference

    AddressLayout layout = null;

    ParamList paramlist = null;

    string sLine = string.Empty;

    try
    {

        ws4 = new InCodeWS4Client();

        paramlist = ws4.GetParamList(NPAR_ParameterType.NPAR_AddressLayouts);

        Console.WriteLine("Adress Layout List:");

        for (int index = 1; index <= paramlist.strParamName.Count; index++)
        {

            sLine = index + ". " + paramlist.strParamName[index - 1];

            Console.WriteLine(sLine);

        }

        Console.WriteLine();

        Console.Write("Enter The Name Of The Adress Layout: ");

        string sLayout = Console.ReadLine();

        layout = ws4.GetLayoutDetails(sLayout);

        if (layout.intRetCode != 0)
        {

            Console.WriteLine();

            Console.WriteLine("Adress Layout Details:");

            Console.WriteLine("Layout Name: " + layout.strLayoutName);

            Console.WriteLine("Total Lines: " + layout.intTotalLines);

            Console.WriteLine("Record Length: " + layout.intRecordLength);

            Console.WriteLine("Address Lines: ");
        }
    }
}

```

```

    for (int index = 1; index <= layout.intTotalLines; index++)
    {
        AddressLine al = layout.Lines[index - 1];

        sLine = index + ". " + al.strLineName;

        sLine += " [" + al.intLineStart + ",";

        int lineend = al.intLineStart + al.intLineLength - 1;

        sLine += lineend + "]";

        // present the address types associated with the line
        for (int subindex = 0; subindex < al.Types.Length; subindex++)
        {
            sLine += " " + al.Types[subindex];
        }

        Console.WriteLine(sLine);
    }
}

else
{
    Console.WriteLine("Error Getting Address Layout Details:");

    Console.WriteLine(layout.strMessage);
}
}

catch (Exception ex)
{
    Console.WriteLine("Exception Caught: " + ex.ToString());
}

Console.ReadLine(); // wait for the user's response
}

```

---

# Address Correction Example in C#.NET

The AddressCorrectionTest() example shows how to perform address correction:

```
static void AddressCorrectionTest()
{
    InCodeWS4Client ws4 = null; // nCodeWS4 proxy reference
    AddressLayout layout = null;
    AnalyzeInput request = new AnalyzeInput();
    AnalyzeOutput response = null;
    string sLine = string.Empty;

    try
    {
        ws4 = new InCodeWS4Client();
        request.AnalysisType = NAA_AnalysisType.ADDRESS_CORRECTION;
        Console.WriteLine("Enter The Name Of The Address Layout: ");
        request.strLayout = Console.ReadLine();
        Console.WriteLine("Enter The Name Of The Analysis Mode: ");
        request.strAnalysis = Console.ReadLine();
        Console.WriteLine("Enter The Name Of The Correction Style: ");
        request.strCorrection = Console.ReadLine();
        // will need the line names for the analysis
        layout = ws4.GetLayoutDetails(request.strLayout);
        if (layout.intRetCode == 0)
        {
            throw new Exception(layout.strMessage);
        }

        request.udtLinesIn = new LineContent[layout.intTotalLines];
    }
```

```

Console.WriteLine();

Console.WriteLine("Enter The Address Line Contents: ");

for (int index = 0; index < layout.intTotalLines; index++)
{
    request.udtLinesIn[index] = new LineContent();
    request.udtLinesIn[index].strName = layout.Lines[index].strLineName;
    Console.Write(layout.Lines[index].strLineName + ": ");
    sLine = Console.ReadLine();
    request.udtLinesIn[index].strContent = sLine;
}

response = ws4.AnalyzeAddress(request); // perform the analysis
if (response.intRetCode != 0)
{
    Console.WriteLine();
    Console.WriteLine("Status: " + response.strStatus);
    Console.WriteLine("Message: " + response.strMessage);
    for (int index = 0; index < response.udtLinesOut.Length; index++)
    {
        sLine = response.udtLinesOut[index].strName + ": ";
        sLine += response.udtLinesOut[index].strContent;
        Console.WriteLine(sLine);
    }
}
else
{
    Console.WriteLine("Error:" + response.strMessage);
}
}

```

```

        catch (Exception ex)
        {
            Console.WriteLine("Exception Caught: " + ex.ToString());
        }

        Console.ReadLine(); // wait for the user's response
    }

```

---

## Address Transformation Example in C#.NET

The AddressTransformationTest() example shows how to perform address transformation:

```

static void AddressTransformationTest()
{
    InCodeWS4Client ws4 = null; // nCodeWS4 proxy reference

    AddressLayout layout = null;

    TransformInput request = new TransformInput();

    TransformOutput response = null;

    string sLine = string.Empty;

    try
    {
        ws4 = new InCodeWS4Client();

        Console.Write("Enter The Name Of The First Adress Layout: ");
        request.strFirstLayout = Console.ReadLine();

        Console.Write("Enter The Name Of The Second Adress Layout: ");
        request.strSecondLayout = Console.ReadLine();

        Console.Write("Enter The Name Of The Analysis Mode: ");
        request.strAnalysis = Console.ReadLine();

        Console.Write("Enter The Name Of The Correction Style: ");
        request.strCorrection = Console.ReadLine();
    }
}

```

```

// will need the line names for the transformation

layout = ws4.GetLayoutDetails(request.strFirstLayout);

if (layout.intRetCode == 0)
{
    throw new Exception(layout.strMessage);
}

request.udtLinesIn = new LineContent[layout.intTotalLines];

Console.WriteLine();

Console.WriteLine("Enter The Address Line Contents: ");

for (int index = 0; index < layout.intTotalLines; index++)
{
    request.udtLinesIn[index] = new LineContent();

    request.udtLinesIn[index].strName = layout.Lines[index].strLineName;

    Console.Write(layout.Lines[index].strLineName + ": ");

    sLine = Console.ReadLine();

    request.udtLinesIn[index].strContent = sLine;
}

response = ws4.TransformAddress(request); // perform the transformation

Console.WriteLine();

Console.WriteLine("Return Code: " + response.intRetCode);

Console.WriteLine("Message: " + response.strMessage);

if (response.intRetCode != 0)
{
    Console.WriteLine();

    Console.WriteLine("Transformed Address:");

    for (int index = 0; index < response.udtLinesOut.Length; index++)
    {
        sLine = response.udtLinesOut[index].strName + ": ";
    }
}

```



```

        sLine += response.udtLinesOut[index].strContent;

        Console.WriteLine(sLine);
    }
}

else
{
    Console.WriteLine("Error:" + response.strMessage);
}

}

catch (Exception ex)
{
    Console.WriteLine("Exception Caught: " + ex.ToString());
}

Console.ReadLine(); // wait for the user's response
}

```

---

## Address Browsing Example in C#.NET

The AddressBrowsingTest() example shows how to perform address browsing:

```

static void AddressBrowsingTest()
{
    InCodeWS4Client ws4 = null; // nCodeWS4 proxy reference

    AddressLayout layout = new AddressLayout();

    BrowseAddressExInput request = new BrowseAddressExInput();

    BrowseAddressExOutput response = null;

    string sLine = string.Empty;

    try
    {

```

```

ws4 = new InCodeWS4Client();

Console.Write("Enter The Name Of The Adress Layout: ");

request.strLayout = Console.ReadLine();

Console.Write("Enter The Name Of The Analysis Mode: ");

request.strAnalysis = Console.ReadLine();

Console.Write("Enter The Name Of The Correction Style: ");

request.strCorrection = Console.ReadLine();

// will need the line names for the browsing
layout = ws4.GetLayoutDetails(request.strLayout);

if (layout.intRetCode == 0)
{
    throw new Exception(layout.strMessage);
}

request.udtLinesIn = new LineContent[layout.intTotalLines];

Console.WriteLine();

Console.WriteLine("Enter The Address Line Contents: ");

for (int index = 0; index < layout.intTotalLines; index++)
{
    request.udtLinesIn[index] = new LineContent();

    request.udtLinesIn[index].strName = layout.Lines[index].strLineName;

    Console.Write(layout.Lines[index].strLineName + ": ");

    sLine = Console.ReadLine();

    request.udtLinesIn[index].strContent = sLine;
}

request.GuessingFlags = new GuessingModes(); // new to nCodeWS4
// for simplicity, use the defaults implied by the analysis mode
request.GuessingFlags.UseDefaults = true;

response = ws4.BrowseAddressEx(request); // perform the browsing

```

```

        Console.WriteLine();

        Console.WriteLine(response.strMessage); // present the outcome

        // if successful, present the address browsing results

        if (response.intRetCode != 0 && response.ResultType !=
DAB_ResultType.DAB_Empty)
        {
            Console.WriteLine();

            if (response.ResultType == DAB_ResultType.DAB_Addresses)
            {
                sLine = "Total " + response.Address.Length + " Addresses
Found.";

                Console.WriteLine(sLine);

                foreach (AddressElements ae in response.Address)
                {
                    sLine = PresentAddress(ae);

                    Console.WriteLine(sLine);
                }
            }

            else // the list of names has been created
            {
                sLine = "Total " + response.Name.Length + " Names Found.";

                Console.WriteLine(sLine);

                foreach (ParserElements pe in response.Name)
                {
                    sLine = PresentPEOneLine(pe);

                    Console.WriteLine(sLine);
                }
            }
        }
    }
}

```

```

    }

    catch (Exception ex)
    {
        Console.WriteLine("Exception Caught: " + ex.ToString());
    }

    Console.ReadLine(); // wait for the user's response
}

// a helper method that formats number + suffix, depending
// on the suffix length ('A' vs. '1/2')
static string NumberSuffix(string number, string suffix)
{
    const string space = " ";
    string ns = String.Empty;

    if (suffix.Length > 1)
    {
        ns = number + space + suffix;
    }
    else
    {
        ns = (number + suffix).Trim();
    }

    return ns;
}

// a helper method that converts the input address into a one-line string
// there are many other ways to improvise with one line address formatting

```

```

static string PresentAddress(AddressElements ae)
{
    string name = String.Empty;
    string mnc = String.Empty;
    string range = String.Empty;
    string lvrintfo = String.Empty;
    string line = String.Empty;
    const string dash = " - ";
    const string comma = ", ";
    const string space = " ";

    // the range point of the address
    switch (ae.TypeOfAddress)
    {
        case AES_TypeOfAddress.AES_StreetAddress:
            // define the street number information
            if (ae.StreetNumberFrom == ae.StreetNumberTo &&
                ae.StreetNumberSuffixFrom == ae.StreetNumberSuffixTo)
            {
                range = NumberSuffix(ae.StreetNumberFrom,
ae.StreetNumberSuffixFrom);
            }
            else
            {
                range = NumberSuffix(ae.StreetNumberFrom,
ae.StreetNumberSuffixFrom) +
                                dash + NumberSuffix(ae.StreetNumberTo,
ae.StreetNumberSuffixTo);
            }

            // append SUITE info, if applicable

```

```

        if (ae.SuiteNumberFrom.Length > 0)
        {
            range += comma + ae.SuiteKeyword;

            if (ae.SuiteNumberFrom == ae.SuiteNumberTo)
                range += space + ae.SuiteNumberFrom;
            else
                range += space + ae.SuiteNumberFrom + dash +
ae.SuiteNumberTo;
        }

        break;

    case AES_TypeOfAddress.AES_SSBRAddress:
        // define the street number information
        if (ae.StreetNumberFrom == ae.StreetNumberTo &&
            ae.StreetNumberSuffixFrom == ae.StreetNumberSuffixTo)
        {
            range = NumberSuffix(ae.StreetNumberFrom,
ae.StreetNumberSuffixFrom);
        }
        else
        {
            range = NumberSuffix(ae.StreetNumberFrom,
ae.StreetNumberSuffixFrom) +
                                dash + NumberSuffix(ae.StreetNumberTo,
ae.StreetNumberSuffixTo);
        }

        // distinguish between odd, even and continuous numbers
        switch (ae.SequenceCode)
        {
            case AES_SequenceCode.AES_ConsecutiveNumbers:
                range += " cons."; break;

```

```

        case AES_SequenceCode.AES_EvenNumbers:

            range += " even"; break;

        case AES_SequenceCode.AES_OddNumbers:

            range += " odd"; break;

        case AES_SequenceCode.AES_SSBRBoxNumbers:

            range += " box"; break;

    }

    // precede the range with the route info
    if (ae.RouteNumber.Length > 0)
    {

        range = ae.RouteKeyword + space + ae.RouteNumber + comma +
range;

    }

    break;

case AES_TypeOfAddress.AES_POBoxAddress:

    range = ae.POBoxKeyword + space;

    if (ae.POBoxNumberFrom == ae.POBoxNumberTo)
    {

        range += ae.POBoxNumberFrom;

    }

    else

    {

        range += ae.POBoxNumberFrom + dash + ae.POBoxNumberTo;

    }

    break;

case AES_TypeOfAddress.AES_RouteAddress:

    range = ae.RouteKeyword + space + ae.RouteNumber;

    break;

case AES_TypeOfAddress.AES_GDAddress:

```

```

        range = ae.GDKeyword;

        break;
    }

    // name portion of the address

    if (ae.TypeOfAddress == AES_TypeOfAddress.AES_StreetAddress ||
        ae.TypeOfAddress == AES_TypeOfAddress.AES_SSBRAddress)
    {
        name = ae.StreetName;

        if (ae.StreetType.Length > 0)
        {
            name += space + ae.StreetType;
        }

        if (ae.StreetDirection.Length > 0)
        {
            name += space + ae.StreetDirection;
        }
    }

    else // station type address
    {
        name = ae.StationName;

        if (ae.StationType.Length > 0)
        {
            name += space + ae.StationType;
        }

        if (ae.StationQualifier.Length > 0)
        {
            name += space + ae.StationQualifier;
        }
    }

```



```

    }

    // municipality, province and postal code portion of the address
    mnc = ae.Municipality + comma + ae.Province + comma + ae.PostalCode;

    // The LVR Information, if applicable
    if (ae.LVRType != AES_TypeOfLVR.AES_NotAnLVR)
    {
        // distinguish between LVR and building names
        if (ae.LVRType == AES_TypeOfLVR.AES_Building)
            lvrinfo = "Building";
        else
            lvrinfo = "LVR";

        // append Government attribute if applicable
        if (ae.LVRType == AES_TypeOfLVR.AES_GovPOBoxLVR || ae.LVRType ==
AES_TypeOfLVR.AES_GovStreetLVR)
            lvrinfo = "GOV. " + lvrinfo;

        lvrinfo += comma;

        // could have also used ae.LVRName, ae.LVRLanguage and ae.BranchName LVR
attributes

        // but that would take another line for presentation
    }

    else // non-LVR
    {
        lvrinfo = String.Empty;
    }

    // the complete line
    line = lvrinfo + range + comma + name + comma + mnc;

    return line;
}

```

```

// a helper method to present ParserElements in one line
static string PresentPEOneLine(ParserElements pe)
{
    string line = String.Empty;
    string name = String.Empty;
    string mnc = String.Empty;
    const string comma = ", ";
    const string space = " ";

    // name portion of the address
    if (pe.StreetName.Length > 0)
    {
        name = pe.StreetName;

        if (pe.StreetType.Length > 0)
        {
            name += space + pe.StreetType;
        }

        if (pe.StreetDirection.Length > 0)
        {
            name += space + pe.StreetDirection;
        }
    }
    else // station type address
    {
        name = pe.StationName;

        if (pe.StationType.Length > 0)
        {
            name += space + pe.StationType;
        }
    }
}

```

```

    }

    if (pe.StationQualifier.Length > 0)
    {
        name += space + pe.StationQualifier;
    }
}

// municipality, province, postal code
mnc = pe.Municipality + comma + pe.Province;

if (pe.PostalCode.Length > 0)
    line += comma + pe.PostalCode;

// the complete line
line = name + comma + mnc;

return line;
}

```

---

## Address Searching Example in C#.NET

The AddressSearchTest() example shows how to perform address searching:

```

static void AddressSearchTest()
{
    InCodeWS4Client ws4 = null; // nCodeWS4 proxy reference

    BrowsingRequest request = new BrowsingRequest();

    AddressesFound response = null;

    string sLine = string.Empty;

    try
    {
        ws4 = new InCodeWS4Client();

        // define the search conditions:

```

```

request.TargetType = NB_TargetType.NB_TargetAddresses;

// for simplicity, search by postal code only
Console.Write("Enter The Postal Code To Search: ");

request.PostalCode = Console.ReadLine();

// set other name conditions to empty
request.Name = string.Empty;
request.Type = string.Empty;
request.Direction = string.Empty;
request.Municipality = string.Empty;
request.Province = string.Empty;
request.NumberFrom = string.Empty;
request.NumberTo = string.Empty;

// set the inclusion conditions
request.IncludeAll = NB_IsIncluded.NB_IncludedNo;
request.AllInRange = NB_AllInRange.NB_AllInRangeYes;
request.IncludeStreet = NB_IsIncluded.NB_IncludedYes;
request.IncludeSSBR = NB_IsIncluded.NB_IncludedYes;
request.IncludePOBox = NB_IsIncluded.NB_IncludedNo;
request.IncludeRoute = NB_IsIncluded.NB_IncludedNo;
request.IncludeGD = NB_IsIncluded.NB_IncludedNo;
request.IncludeUrban = NB_IsIncluded.NB_IncludedYes;
request.IncludeRural = NB_IsIncluded.NB_IncludedYes;

// set the conditions that are new to nCodeWS4
request.ApplyMisspellings = false;
request.ApplyMissingWords = false;
request.AccentedInput = true;
request.AccentedOutput = true;
request.LowercaseOutput = true;

```

```

        // perform the search

        response = ws4.SearchForAddresses(request);

        Console.WriteLine();

        if (response.intRetCode == 0) // failed, report the error
        {
            Console.WriteLine("Error:" + response.strMessage);
        }

        else // present the results
        {
            sLine = "Total " + response.Address.Length + " Addresses Found.";
            Console.WriteLine(sLine);

            foreach (AddressElements ae in response.Address)
            {
                sLine = PresentAddress(ae);
                Console.WriteLine(sLine);
            }
        }
    }

    catch (Exception ex)
    {
        Console.WriteLine("Exception Caught: " + ex.ToString());
    }

    Console.ReadLine(); // wait for the user's response
}

```



# nCodeWS4 Methods Reference

## nCodeWS4 Methods

The methods provided by nCodeWS4 and nCodeWS4R Web Services are listed here. nCodeWS4R Web Service does not implement all of nCodeWS4 methods, only the most commonly used ones (see [nCodeWS4R Web Service](#)). For each method, its usage, its method signature, the request elements, the response elements and the rules that apply to the request and the response are provided. The following methods are new to nCodeWS4: [SearchForAltMunicipalities](#), [SearchForAltStreets](#) and [OneStopAnalyze](#).

All nCodeWS4 methods belong to "http://novamg.com/nCodeWS4/" namespace.

Methods for address analysis, address transformation and address formatting:

[AnalyzeAddress](#)

[AnalyzeGetAlternatives](#)

[AnalyzeRecords](#)

[TransformAddress](#)

[FormatAddress](#)

[FormatPE](#)

Methods for address database searches:

[SearchForAddresses](#)

[SearchForNames](#)

[SearchForAltMunicipalities](#)

[SearchForAltStreets](#)

[GetKeywords](#)

Methods for address parsing, address browsing and address capture:

[ParseAddress](#)

[BrowseAddressEx](#)

[OneStopAnalyze](#)

Methods for managing address layouts, analysis modes and correction styles:

[GetParamList](#)

[RemoveParameter](#)

[GetLayoutDetails](#)

[GetAnalysisMode](#)

[GetCorrectionStyle](#)

[SetAddressLayout](#)

[SetAnalysisMode](#)

[SetCorrectionStyle](#)

Method for getting detailed information about the Web Service:

[GetWebServiceInfo](#)

---

## AnalyzeAddress Method

The AnalyzeAddress method is used for address validation, address correction and postal code lookup. This method implements a strict set of rules for address validation and correction imposed by Canada Post. The AnalyzeAddress method has the following signature:

[AnalyzeOutput](#) **AnalyzeAddress**([AnalyzeInput](#) request);

The request consists of:

- The name of the [address layout](#) to be used in address analysis
- The name of the [analysis mode](#) to be applied during the address analysis
- The name of the [correction style](#) to be used when correcting/optimizing the address
- The type of the address analysis, which can be either address validation, address correction or postal code lookup
- The content of the address record to be analyzed, represented as a series of [address line](#) name/address line content pairs

The response consists of:

- The content of the analyzed address record, represented as a series of [address line](#) name/address line content
- The [status code](#) of the analyzed address record
- The number of alternatives found during the analysis
- The number that uniquely identifies the address record in the nCode Address Reference Database. Note that this number will change from month to month for any given address range.
- [Address style changes](#) that are result of address correction and optimization
- The hash code associated with the input address, produced by the 64 bit FNV hashing algorithm. For valid and correctable addresses, the hash is calculated against the normalized database address associated with the input address. For non-correctable addresses, the hash is calculated against the normalized input address. The calculated hash value is independent of nCode database version and it can be used to identify records that are variations of the same address.
- Foreign country name, in case that the address analyzed was declared foreign. If the address was not declared foreign, define the country as CANADA.
- The return code. Zero represents failure and one represents success
- The one-line message that describes the result of the analysis

Please note the following:



- For non-correctable addresses, this method returns only the number of alternatives. In order to get the actual alternatives themselves, the [AnalyzeGetAlternatives](#) method should be used.
- Address style changes reported are new to nCodeWS4. For backwards compatibility purposes, they can be ignored. Alternatively, they can be used to enhance reporting on the results of address analysis.

---

## AnalyzeGetAlternatives Method

The [AnalyzeGetAlternatives](#) method is used for those situations when you want to get the details of the address alternatives that nCodeWS3 finds when determining the status of a given input address. This method is most useful for following up on addresses that are deemed to be “non-correctable with alternatives”, it can also be used for valid and correctable addresses. The key advantages of the [AnalyzeGetAlternatives](#) method over the [AnalyzeAddress](#) method are:

- Access to all address elements related to a particular address alternative.
- Access to detailed messages related to a particular address alternative.
- Messages are provided not only in English, but also through machine-readable message codes, so that messages can be generated in any language.
- When you want to transform addresses from one input address layout to a different output address layout, using [AnalyzeGetAlternatives](#) saves one step by giving you an address element structure that can be passed to the [FormatAddress](#) method.

Since the number of address alternatives can be large, the [AnalyzeGetAlternatives](#) method can be somewhat slower than the [AnalyzeAddress](#) method. Also, the [AnalyzeGetAlternatives](#) method assumes automatic address correction; therefore it cannot be used for address validation or postal code lookup.

The [AnalyzeGetAlternatives](#) method has the following signature:

[AnalyzeGetAlternativesOutput](#) [AnalyzeGetAlternatives](#)([AnalyzeGetAlternativesInput](#) request);

The request consists of:

- The name of the [address layout](#) to be used in address analysis
- The name of the [analysis mode](#) to be applied during the address analysis
- The name of the [correction style](#) to be used when correcting/optimizing the address
- The content of the address record to be analyzed, represented as a series of [address line](#) name/address line content pairs

The response consists of:

- The content of the analyzed address record, represented as a series of [address line](#) name/address line content.
- The [status code](#) of the analyzed address record.
- The number of alternatives found during the analysis.
- The number that uniquely identifies the address record in the nCode Address Reference Database. Note that this number will change from month to month for any given address range.
- The series of [address alternatives](#) found during the address analysis. Each alternative includes both the address elements of the address alternative and the messages related to this alternative.
- [Address style changes](#) that are result of address correction and optimization.
- The hash code associated with the input address, produced by the 64 bit FNV hashing algorithm. For valid and correctable addresses, the hash is calculated against the normalized database address associated with the input address. For non-correctable addresses, the hash is calculated against the normalized input address. The

calculated hash value is independent of nCode database version and it can be used to identify records that are variations of the same address.

- Foreign country name, in case the address analyzed was declared foreign. If the address was not declared foreign, define the country as CANADA.
- The return code. Zero represents failure and one represents success.
- The one-line message that describes the result of the analysis.

Please note the following:

- The messages related to a particular address alternative offer a fine level of detail, including the following: address type to which they relate, if applicable; the correct value of the address type, if applicable; the error category of the message (major error, minor error, etc.); the subcategory of the message (address type was missing, misspelled, etc.); and the text equivalent of the message in English.
- Address style changes reported are new to nCodeWS4. For backwards compatibility purposes, they can be ignored. Alternatively, they can be used to enhance reporting on the results of address analysis.

---

## AnalyzeRecords

The `AnalyzeRecords` method was added in order to assist processing of address records in a batch mode. Instead of analyzing address records one at a time, as is the case with `AnalyzeAddress` and `AnalyzeGetAlternatives` methods, `AnalyzeRecords` accepts a list of address records, which are then processed all at once. This approach can result in processing that is up to five times faster, when compared to invoking `AnalyzeGetAlternatives` for every single address record. Client applications can utilize `AnalyzeRecords` method only in PCAD data context. The PoCAD data context is reserved for SOA generation scenario, which can be done by a dedicated Nova application only. For that purpose, Nova will provide “nCodeWS4 Batch” application that will be able to do SOA generation against nCodeWS4 implementations, including nCodeServer4, nCodeServer4J and nCodeServer4 pay-per-use versions. The `AnalyzeRecords` method was introduced with nCodeWS4 version 4.7.

The `AnalyzeRecords` method has the following signature:

[AnalyzeRecordsOutput](#) `AnalyzeRecords`([AnalyzeRecordsInput](#) request);

The request consists of:

- The name of the [address layout](#) to be used in address analysis
- The name of the [analysis mode](#) to be applied during the address analysis
- The name of the [correction style](#) to be used when correcting/optimizing the address
- The type of the address analysis, which can be either address validation, address correction or postal code lookup
- The list of address records to be analyzed
- The flag which determines whether alternative messages should be generated

The response consists of:

- The list of output address records and their analysis details
- Data context under which address analysis was done (PCAD=1 or PoCAD=2)
- Success status of the method call
- Description of the outcome of the method call

The AnalyzeRecords method limits the maximum size of the input address records list to 1024.

---

## TransformAddress Method

The TransformAddress method is used when there is a need to convert addresses from one address layout to another. For valid and correctable addresses the address transformation should always be successful, unless the fields of the resulting address layout are too small to receive the data. For non-correctable addresses with at least one alternative address, the address transformation should always be partially successful, which means that only those address elements that are shared by all alternative addresses will be carried on to the resulting address. In the case of a valid or a correctable address, the resulting address will be corrected by using the correction style provided with the request. In the case of a non-correctable address with alternatives, the address elements in the resulting address will not be optimized. In the case of a non-correctable address with no alternatives or a foreign address, the address transformation will fail completely. The TransformAddress method has the following signature:

[TransformOutput](#) TransformAddress([TransformInput](#) request);

The request consists of:

- The name of the originating [address layout](#) to be used in address transformation
- The name of the resulting [address layout](#) to be used in address transformation
- The name of the [analysis mode](#) to be applied during address transformation
- The name of the [correction style](#) to be used in address transformation
- The content of the address record to be transformed, represented as a series of [address line](#) name/address line content pairs

The response consists of:

- The content of the transformed address record, represented as a series of [address line](#) name/address line content pairs
- The return code with possible values from zero to nine representing different outcomes of the address transformation as follows:
  - 0 = Call failed
  - 1 = Successful
  - 2 = Input record was null
  - 3 = Transformation not set properly
  - 4 = Cannot transform foreign word
  - 5 = Cannot transform a non-correctable address with no alternatives
  - 6 = Partial transformation for a non-correctable address with alternatives
  - 7 = One of the output address fields was too small for the data
  - 8 = Transformed record is not valid
  - 9 = One of the output non-address fields was too small for the data
- The one-line message that describes the result of the address transformation

---

## FormatAddress Method

The FormatAddress method is used to format a set of address elements retrieved from the nCode Address Reference Database into an address record. Since most addresses in the nCode Address Reference Database contain ranges, an

actual number in the address range that uniquely defines the resulting address record is usually also required. This method can also be used when a number of address alternatives are obtained by calling the [AnalyzeGetAlternatives](#) method and the end user has selected a particular address alternative to be formatted. The FormatAddress method has the following signature:

[FormatAddressOutput](#) **FormatAddress**([FormatAddressInput](#) request);

The request consists of:

- The name of the [address layout](#) to be used in address formatting
- The name of the [analysis mode](#) to be used in address formatting
- The name of the [correction style](#) to be used in address formatting
- The [address elements](#) representing the address range
- The actual number in the range to be used in address formatting (if applicable)

The response consists of:

- The content of the formatted address record, represented as a series of [address line](#) name/address line content pairs
- The return code. Zero represents failure and one represents success.
- The one-line message that describes the result of the formatting

Please note that in order to avoid surprises, the address elements input to this method should be those obtained from other methods, such as the [AnalyzeGetAlternatives](#) or [SearchForAddresses](#) methods. It is possible to define the address elements manually from scratch, but in this case one must be careful not to omit any address elements or introduce inconsistent address elements, since this method does not perform address analysis or correction.

---

## FormatPE Method

The purpose of the FormatPE method is to transform a [ParserElements](#) structure into an address record in a given address layout. Unlike the [FormatAddress](#) method, which formats the final address record and applies correction styles to the output, the FormatPE method merely arranges the address elements given in a ParserElements structure according to the given address layout without applying correction styles to the output. This method is useful for direct address browsing ([BrowseAddressEx](#) method) when a browsing attempt returns a list of names instead of a list of address ranges. In such cases, the user can be offered a list of ParserElements structures from which to choose. The user's selection can be fed to the FormatPE method to convert it into an address record suitable for use as an input to a subsequent call to BrowseAddressEx. In this way, a user can drill down from a street/station name search to an address range search. The FormatPE method has the following signature:

[FormatPEOutput](#) **FormatPE**([FormatPEInput](#) request);

The request consists of:

- The name of the [address layout](#) to be used in address formatting
- The name of the [analysis mode](#) to be used in address formatting
- The name of the [correction style](#) to be used in address formatting
- The [ParserElements](#) structure

The response consists of:

- The content of the formatted address record, represented as a series of address line name/address line content pairs
- The return code. Zero represents failure and one represents success.

- The one-line message that describes the result of the formatting

---

## SearchForAddresses Method

The SearchForAddresses method enables the user to perform conditional searches for addresses against the nCode Address Reference Database, based on a wide range of conditions. The SearchForAddresses method has the following signature:

[AddressesFound](#) SearchForAddresses([BrowsingRequest](#) request);

The request represents the search conditions (filter criteria), which include the following subgroups:

Name conditions apply to:

- street/station name
- street/station type
- street direction / station qualifier
- municipality name
- province name

Range conditions include:

- number from
- number to
- whether the address records found should fall into the from/to range
- postal code / FSA condition

Address type conditions include:

- include all address types
- include street addresses
- include street served by route addresses
- include PO Box addresses
- include rural route addresses
- include general delivery addresses
- include urban addresses
- include rural addresses

The following conditions are new to nCodeWS4 and are used to explicitly define those search conditions and properties that were previously determined implicitly through the underlying analysis modes and correction styles:

- include misspelled street and station names
- include street and station names missing words
- expect accented input
- produce accented output
- produce lowercase output

These rules apply to the search conditions:

- When a name condition is empty or missing it indicates that any name found will satisfy this particular condition.
- When the number from condition is empty or missing it indicates that any address record will satisfy this condition.
- When the number to condition is empty or missing it indicates that any address record will satisfy this condition.
- When include all address type indicator is set to yes, it overrides all other address type indicators, therefore any address record type will satisfy this condition.
- Only the address records that satisfy all address search conditions will be presented.

The response consists of:

- An array of the addresses found. Each address is represented by a complex structure, which includes all address details, such as the type of address, name elements, range elements, etc.
- The return code. Zero represents failure and one represents success
- The message that describes the result of the address search

---

## SearchForNames Method

The SearchForNames method enables you to perform conditional searches for street/station names, municipality names or FSA names against the nCode Address Reference Database, based on a wide range of conditions. The SearchForNames method has the following signature:

[NamesFound](#) SearchForNames([BrowsingRequest](#) request);

For a description of the request, please see the [SearchForAddresses](#) method. The only difference here is that while the Target Type element of the request is fixed to target addresses in the SearchForAddresses case, the Target Type in the case of SearchForNames can indicate names, municipalities or FSAs.

The response consists of:

- The array of the names found. Each name found is represented by a structure, which includes street/station name, street/station type, street direction/station qualifier, municipality, province and FSA.
- The return code. Zero represents failure and one represents success.
- The message that describes the result of the name search.

The following rules apply to name searches:

- If the Target Type is search names, the resulting structures will contain the street/station name, street/station type, street direction/station qualifier, municipality and province.
- If the Target Type is search municipalities, the resulting structures will contain the municipality and province.
- If the Target Type is search FSAs, the resulting structures will contain the FSAs only.

---

## SearchForAltMunicipalities Method

The SearchForAltMunicipalities method enables you to perform conditional searches for alternate municipality names against the nCode Address Reference Database. Both the alternate municipality search conditions and the alternate municipality records retrieved from the database are represented with [AltMncRecord](#) structure. The SearchForAltMunicipalities method is new to nCodeWS4 and it has the following signature:

[AltMncSearchOutput](#) SearchForAltMunicipalities([AltMncSearchInput](#) request);

The request consists of the alternate municipality search conditions, given in the form of [AltMncRecord](#) structure:

- The official municipality name
- The alternate municipality name
- The type of alternate municipality name (valid, invalid, 13 character abbreviation, 18 character abbreviation or any type)
- The alternate municipality name validity indicator (acceptable, unacceptable or any validity)
- The FSA condition
- The province condition

These rules apply to the search conditions:

- When a name condition is empty or missing it indicates that any name found will satisfy this particular condition.
- When any type of alternate municipality name is used as a condition, any alternate municipality record will satisfy this particular condition.
- When any alternate municipality name validity indicator is used as a condition, any alternate municipality record will satisfy this particular condition.
- The FSA condition is used to further narrow validity of alternate municipality names to only those addresses that belong to the given forward sortation area (FSA).
- The province condition states that the alternate municipality name relates to the official municipality name within the given province only.

The response consists of:

- The [AltMncSearchResult](#) based return code. The value Successful represents success and other values represent failure.
- The array of the alternate municipality records found. Each record is given as [AltMncRecord](#) structure and consists of municipality name, alternate municipality name, type of alternate municipality name, validity indicator, FSA condition and province condition.
- The message that describes the result of the search.

Please note that:

- The resulting array of the alternate municipality records could be empty.
- If the type of alternate municipality name was Invalid or if the alternate municipality name validity indicator was Unacceptable, the alternate municipality name cannot be used in valid addresses (conditional on the province and the FSA). nCodeWS4 recognizes unacceptable alternate municipality names and corrects them, if otherwise possible.
- If invalid search conditions were given, the return code will state which particular search condition was invalid.

---

## SearchForAltStreets Method

The SearchForAltStreets method enables you to perform conditional searches for alternate street names against the nCode Address Reference Database. Both the alternate street search conditions and the alternate street records retrieved from the database are represented with [AltStrRecord](#) structure. The SearchForAltStreets method is new to nCodeWS4 and it has the following signature:

[AltStrSearchOutput](#) SearchForAltStreets([AltStrSearchInput](#) request);

The request consists of the alternate street search conditions, given in the form of [AltStrRecord](#) structure:

- The official street name
- The official street type
- The official street direction
- The alternate street name
- The alternate street type
- The alternate street direction
- The municipality condition
- The province condition

Additionally, the request consists of the following flags indicating whether empty street type and empty street direction conditions are explicit:

- Is empty street type explicit flag
- Is empty street direction explicit flag
- Is empty alternate street type explicit flag
- Is empty alternate street direction explicit flag

These rules apply to the search conditions:

- When a name condition is empty or missing it indicates that any name found will satisfy this particular condition.
- The municipality and the province define the conditions under which the alternate street name, alternate street type and alternate street direction represent a valid alternative to the street name, street type and street direction.
- Street names are sometimes missing their street types and/or street directions. Therefore, when providing alternate street search conditions, we need to specify whether empty street type and empty street direction are explicitly empty or any value in their place is allowed.

The response consists of:

- The [AltStrSearchResult](#) based return code. The value Successful represents success and other values represent failure.
- The array of the alternate street records found. Each record is given as [AltStrRecord](#) structure and consists of street name, type and direction, alternate street name, type and direction, municipality condition and province condition.
- The message that describes the result of the search.

Please note that:

- The resulting array of the alternate street records could be empty.
- The alternate street name, type and direction can be used in place of the street name, type and direction in valid addresses (conditional on the municipality and the province).
- If invalid search conditions were given, the return code will state which particular search condition was invalid.

---

## GetKeywords Method

The GetKeywords method provides a list of Canada Post defined keywords. The GetKeywords method has the following signature:

[GetKeywordsOutput](#) GetKeywords([KWRequest](#) request);



The request consists of three parameters:

- Address type. Not all address types are associated with keywords. Only street type, street direction, station type, unit type, route keyword, Po Box keyword, GD keyword and province are associated with predefined keywords.
- Language indicator: can be English, French or Both.
- Optimum indicator indicates whether all values should be returned or just the optimum (Canada Post standard) ones.

The response consists of:

- The list of keywords found. Note that this list can be empty.
- The return code. Zero represents failure and one represents success.
- The message that describes the result of the method.

---

## ParseAddress Method

While address validation, correction and postal code lookup are based on a strict set of rules imposed by Canada Post, there are situations when we are just interested in finding address elements in an input address, regardless of how many address elements were provided or whether such an address makes sense semantically. For example, as part of a broadly based search operation, we could parse an address and feed the elements found into the native search methods ([SearchForAddresses](#) or [SearchForNames](#)). It is clear that some input addresses may be interpreted in several ways. Therefore, the ParseAddress method will return an array of [ParserElements](#) structures, that is, structures containing elements of the parsed address. Please note that due to the way that the nCode engine is implemented, duplicates in the resulting array of ParserElements structures are possible. The ParseAddress method has the following signature:

[ParseAddressOutput](#) **ParseAddress**([ParseAddressInput](#) request);

The request consists of:

- The name of the [address layout](#) to be used in parsing.
- The name of the [analysis mode](#) to be used in parsing.
- The name of the [correction style](#) to be used in parsing.
- The [guessing modes](#) to be used in parsing.
- The content of the address record to be parsed, represented as a series of [address line](#) name/address line content pairs

The response consists of:

- The resulting array of [ParserElements](#) structures; the array can be empty.
- The return code. Zero represents failure and one represents success.
- The one-line message that describes the result of the parsing.
- The resulting array of addresses found counters, matching the resulting array of [ParserElements](#) structures by the array index.

Please note the following:

- The names of the correction style and the analysis mode are needed, as the implementation of this method is also based on the address analysis algorithm.
- The guessing modes are new to nCodeWS4 and are used in conjunction with the analysis mode to further control the parsing process.

- The addresses found counters were introduced with nCodeWS4 version 4.3. The idea is to sort the resulting array of [ParserElements](#) structures by the number of matching database address records, so that the first [ParserElements](#) structure listed is the most relevant.
- In order to define the addresses found counters, ParseAddress runs additional queries in the nCodeWS4 version 4.3 and later, which can make it significantly slower than the versions prior to the version 4.3, depending on the number of parser variants found. In general, we can lower the number of parser variants found by disabling guessing modes explicitly.

---

## BrowseAddressEx Method

The BrowseAddressEx method is normally used in situations where the address record lacks too many elements. In such cases, the regular address analysis method may not provide the expected alternatives, since the address analysis process is bound by the Canada Post rules. The BrowseAddressEx method will then find the closest matches for the address elements in the address record.

In a typical scenario for direct address browsing, these steps would be the following:

- Invoke the BrowseAddressEx method, passing it the input address record.
- Examine the type of the result. nCode will return either a list of names (streets/stations) or a list of address ranges.
- If a list of names is returned, the user can be presented with this list and asked to select one of the alternatives. When the user selects a name from the list, the [FormatPE](#) method is invoked to generate an intermediate address record. This address record is fed into the BrowseAddressEx method to generate the list of addresses. Note that if there is only one element in the list of names, nCode will skip the initial return and automatically invoke FormatPE and BrowseAddressEx.
- If a list of address ranges is returned, the user can be presented with this list and asked to select one of the alternatives. Once the user selects an address from the list, invoke the [FormatAddress](#) method to generate the final address record. If the selected address contained a range of numbers, the user may be prompted to specify the exact number. If the input address record already contains the number (the [ParserElements](#) structure is consulted), then the number is included automatically. If there is only one address range returned in the list of address ranges, nCode will skip the initial list return and invoke the FormatAddress method automatically.

The BrowseAddressEx method has the following signature:

[BrowseAddressExOutput](#) BrowseAddressEx([BrowseAddressExInput](#) request);

The request consists of:

- The name of the [address layout](#) to be used in direct address browsing.
- The name of the [analysis mode](#) to be applied during direct address browsing.
- The name of the [correction style](#) to be used in direct address browsing.
- The [guessing modes](#) to be used in direct address browsing.
- The content of the address record to be searched for, represented as a series of [address line](#) name/address line content pairs.

The response consists of:

- The type of the result, which can indicate addresses, names or empty result.
- The array of [ParserElements](#) structures describing parser variants. The array can be empty.
- The array of [AddressElements](#) structures describing addresses found. The array can be empty.

- The array of [ParserElements](#) structures describing names found. The array can be empty.
- The array of indexes, linking names or addresses found with a particular parser variant.
- The return code. Zero represents failure and one represents success.
- The one-line message that describes the result of the direct address browsing.

Please note the following:

- The names of the correction style and the analysis mode are still needed as the implementation of this method is also based on the address analysis algorithm.
- The guessing modes are new to nCodeWS4 and are used in conjunction with the analysis mode to further control direct address browsing.

---

## OneStopAnalyze Method

The OneStopAnalyze method is used in the address capture process. The main idea is to get to the targeted address with as little typing as possible and in as few steps as possible. As the user types, the nCodeWS4 client application keeps invoking the OneStopAnalyze method until it arrives at the complete address. The alternatives provided by each method call can be used to help the user narrow down the choices. The user can start from anywhere in the address. For example, the user can start by entering street number and street name first or by entering postal codes first, etc.

The OneStopAnalyze method has the following signature:

[OneStopAnalyzeOutput](#) OneStopAnalyze([OneStopAnalyzeInput](#) request);

The request consists of:

- The name of the [address layout](#) to be used in address capture
- The name of the [analysis mode](#) to be used in address capture
- The name of the [correction style](#) to be used in address capture
- The [guessing modes](#) to be used in address capture
- The content of the address record to be analyzed, represented as a series of [address line](#) name/address line content pairs
- The selected alternative
- The maximum number of alternatives in the response

The following rules apply to the request :

- The source for the selected alternative is the list of alternatives obtained from a previous OneStopAnalyze method call. Therefore, the selected alternative provided in the first OneStopAnalyze method call, in a series of address capture calls, should be empty.
- The selected alternative must be passed exactly as originally provided, otherwise, the parsing of the selected alternative may fail. If no alternative was selected, the selected alternative should be empty.
- The maximum number of alternatives is there to limit the number of alternatives provided in the response to the OneStopAnalyze method call. If the value passed is zero, the number of alternatives provided in the response is unlimited.

The response consists of:

- The success status of the method call indicator (true or false)
- The address complete indicator (true or false)

- The content of the formatted address, represented as a series of [address line](#) name/address line content pairs
- The content of the suggested address, represented as a series of [address line](#) name/address line content pairs
- The address element details in the form of [AddressElements](#) structure
- Type of the list of alternatives (names, addresses or empty list)
- The number of the actual alternatives found
- The list of alternatives
- The message that describes the result of the method
- The data source of the resulting address

The following rules apply to the response:

- If the success status of the method call is false, the message that describes the result of the method will contain the related error message and other response elements will not be relevant.
- If the address complete indicator is true, the address capture process is complete and a fully formatted address, its address elements and its data source are provided.
- The address elements array will always have either just one element (when either the address complete indicator is true or the suggested address is provided) or it will be empty.
- If the address complete indicator is false and the list of alternatives contains only one alternative, it means that the address capture process has narrowed down the choices to a single address record containing ranges. In this case, the suggested address will contain a formatted address with a number placeholder (?###) in place of a missing number. The placeholder then can be used to prompt the user to enter a specific number from the range and then invoke the method again.
- If the address complete indicator is false and the list of alternatives contains multiple elements, we could prompt the user to select an alternative from the list of alternatives, and if that happens, invoke the method again, passing the selected alternative.
- The list of alternatives can be used as a list of suggestions, so that the user keeps typing based on the suggestions presented, without actually selecting an alternative. With more content typed in, we can invoke the method again.
- If the address complete indicator is false and the list of alternatives contains no alternatives, it means that either insufficient or inconsistent address content has been provided.
- The number of the actual alternatives found tells us how many alternatives were actually found during the OneStopAnalyze method call. If this number exceeded the maximum number of alternatives passed in the request, not all of the alternatives found would be present in the list of alternatives.

---

## GetParamList Method

The purpose of the GetParamList method is to retrieve lists of the names of existing [address layouts](#), [analysis modes](#) or [correction styles](#). The names of address layouts, analysis modes or correction styles must be provided to many other methods, like [AnalyzeAddress](#) or [TransformAddress](#). This method is most useful when:

- You want to verify that a certain address layout, analysis mode or correction style already exists.
- You want to present the end user with a choice of existing address layouts, analysis modes or correction styles, so that they can dynamically choose or change the settings.

The GetParamList method has the following signature:

[ParamList](#) GetParamList([NPAR\\_ParameterType](#) request);

The request consists of:

- The type of the parameter list (address layouts, analysis modes or correction styles).

The response consists of:

- The array of strings representing the names of address layouts, analysis modes or correction styles.
- The return code. Zero represents failure and one represents success.
- The message that describes the result of the method.

This method is not needed for situations where the names of the address layouts, analysis modes and correction styles that will be used are already known at design time.

**Note:** The names of the address layouts, analysis modes and correction styles are case sensitive.

---

## GetParamListEx Method

The purpose of the GetParamListEx method is to retrieve lists of the names of existing address layouts, analysis modes or correction styles for a specific owner. The names of address layouts, analysis modes or correction styles must be provided to many other methods, like AnalyzeAddress or TransformAddress. This method is most useful when:

- You want to verify that a certain address layout, analysis mode or correction style already exists for a given owner or all owners.
- You want to present the end user with a choice of existing address layouts, analysis modes or correction styles, so that they can dynamically choose or change the settings for a given owner or all owners.

The GetParamListEx method has the following signature:

[ParamList](#) GetParamListEx([GetParamRequest](#) request);

The request consists of:

- The type of the parameter list (address layouts, analysis modes or correction styles).
- The owner of the parameters in the list (address layouts, analysis modes or correction styles).

The response consists of:

- The array of strings representing the names of address layouts, analysis modes or correction styles.
- The return code. Zero represents failure and one represents success.
- The message that describes the result of the method.

This method is not needed for situations where the names of the address layouts, analysis modes and correction styles that will be used are already known at design time.

Note: The names of the address layouts, analysis modes and correction styles are case sensitive. The GetParamListEx method was added with nCode 12.6.

---

## RemoveParameter Method

The RemoveParameter method is used on those rare occasions when some of the existing [address layouts](#), [correction styles](#) or [analysis modes](#) are not needed any more. Calls to the RemoveParameter method include the name of the address layout, correction style or the analysis mode to be permanently removed from the nCode maintained repository.

This method should be used with great caution, since it may cause the loss of information and disrupt users that are dependent on using the address layout, correction style or the analysis mode being removed. You may wish to perform a backup of the nCode Server files before using this method.

The advantages of maintaining short lists of address layouts, correction styles or analysis modes are:

- Slight improvement in performance, since the lists are traversed sequentially
- Avoiding name collisions and misinterpretations based on having similarly named address layouts, correction styles or analysis modes

The RemoveParameter method has the following signature:

[OperationResult](#) **RemoveParameter**([RemoveParamRequest](#) request);

The request consists of:

- The type of the parameter list (address layouts, analysis modes or correction styles)
- The name of the address layout, analysis mode or correction style to be removed from the currently defined list

The response consists of:

- The return code. Zero represents failure and one represents success.
- The message that describes the result of the method.

---

## GetLayoutDetails Method

The GetLayoutDetails method provides a way to view all of the details of a particular [address layout](#). The GetLayoutDetails method has the following signature:

[AddressLayout](#) **GetLayoutDetails**(String request);

The request consists of the name of the address layout from which details are to be retrieved.

The response consists of:

- The name of the address layout.
- The length of the address layout.
- The number of address lines in the address layout.
- An array of [address line structures](#), each of which include the address line name, the start and the end of the address line and an array of address types associated with the address line.
- The return code. Zero represents failure and one represents success.
- The message that describes the result of the method.

This method is very useful in cases where it is preferable to retrieve address layout details dynamically rather than hard coding them. This method is also very useful for situations where the user can dynamically switch from one layout to another. In such cases, only one common routine is needed to handle all address layouts. Please note that the names of the address layouts are case sensitive.

---

## GetAnalysisMode Method

The GetAnalysisMode method provides a way to view all of the details of a particular [analysis mode](#).

The GetAnalysisMode method has the following signature:

[AnalysisModeOutput](#) **GetAnalysisMode**(String request);

The request consists of the name of the analysis mode from which details are to be retrieved.

The response consists of:

- The name of the analysis mode
- Whether alternate words will be used in the analysis
- Whether the input may be accented
- The maximum number of changed key address elements
- The spelling level of the analysis
- Whether guessing of missing words in street and station names should be attempted
- Whether missing route information in rural street served by route addresses should be penalized
- Which address types are the key address elements
- Which address types must be separated by blanks
- Which address types cannot be changed
- The default guessing modes associated with this analysis mode
- Can the analysis mode be used to generate statement of accuracy (SOA)
- The reason why the analysis mode may not be SOA compliant
- The return code. Zero represents failure and one represents success
- The message that describes the result of the method

This method will be used rarely, mostly to verify that the proper values are associated with the name of a particular analysis mode. For general use, the default analysis mode: 'MODERATE' should be used. Please note that the names of the analysis modes are case sensitive.

---

## GetCorrectionStyle Method

The GetCorrectionStyle method provides a way to view the details of a particular [correction style](#).

The GetCorrectionStyle method has the following signature:

[CorrectionStyleOutput](#) **GetCorrectionStyle**(String request);

The request consists of the name of the correction style to be retrieved.

The response consists of:

- The name of the correction style.
- The letter case formatting style.
- The accented letters formatting style.
- The unit information formatting style.
- The punctuation handling style.

- The postal code form style.
- The extra information handling style.
- The push lines style.
- The street name optimization style.
- The municipality name optimization style.
- How misspellings should be handled for certain address types.
- How certain address types should be optimized.
- Can the correction style be used to generate statement of accuracy (SOA).
- The reason why the correction style may not be SOA compliant.
- The return code. Zero represents failure and one represents success.
- The message that describes the result of the method.

The address types the response relates to are: street type, station type, street direction, unit keyword, PO Box keyword, route keyword, general delivery keyword and province. These are the address types for which Canada Post has a prescribed list of official values.

This method will be used rarely, mostly to verify that the proper values are associated with the name of a particular correction style. For general use, the 'MIXED AS IS' correction style will be sufficient. Please note that the names of correction styles are case sensitive.

---

## SetAddressLayout Method

The SetAddressLayout method provides a way to both create new [address layouts](#) and to redefine existing address layouts. The SetAddressLayout method has the following signature:

[OperationResult](#) SetAddressLayout([NewAddressLayout](#) request);

The request contains all the address layout details, which include:

- The name of the address layout.
- The length of the address layout.
- The number of address lines in the address layout.
- The array of [address line](#) structures, which include the address line names, the start and the end of each address line and the array of address types associated with each address line.

The response consists of:

- The return code. Zero represents failure and one represents success.
- The message that describes the result of the method.

---

## SetAnalysisMode Method

The SetAnalysisMode method provides a way to both create new [analysis modes](#) and to redefine existing analysis modes. The SetAnalysisMode method has the following signature:

[OperationResult](#) SetAnalysisMode([AnalysisModeInput](#) request);

The request contains all the analysis mode details, which include:



- The name of the analysis mode
- Whether alternate words will be used in the analysis
- Whether the input may be accented
- The maximum number of changed key address elements
- The spelling level of the analysis
- Whether guessing of missing words in street and station names should be attempted
- Whether missing route information in rural street served by route addresses should be penalized
- Which address types are the key address elements
- Which address types must be separated by blanks
- Which address types cannot be changed

The response consists of:

- The return code. Zero represents failure and one represents success.
- The message that describes the result of the method.

---

## SetCorrectionStyle Method

The SetCorrectionStyle method provides a way to both create new [correction styles](#) and to redefine existing correction styles. The SetCorrectionStyle method has the following signature:

[OperationResult](#) **SetCorrectionStyle**([NCSCorrectionStyle](#) request);

The request contains all the correction style details, which include:

- The name of the correction style
- The letter case formatting style
- The accented letters formatting style
- The unit information formatting style
- The punctuation handling style
- The postal code form style
- The extra information handling style
- The push lines style
- The street name optimization style
- The municipality name optimization style
- How misspellings should be handled for certain address types
- How certain address types should be optimized

The address types the request relates to are: street type, station type, street direction, unit keyword, PO Box keyword, route keyword, general delivery keyword and province. These are the address types for which Canada Post has a prescribed list of values.

The response consists of:

- The return code. Zero represents failure and one represents success.

- The message that describes the result of the method.

---

## GetWebServiceInfo Method

The purpose of the GetWebServiceInfo method is to retrieve version information about the Web Service. The method provides detailed information about the version of the Web Service WSDL and its implementation, native nCode library version and the address data version it references, as well as the information about the runtime environment, including the Web Server application name and version, underlying virtual machine name and version and the operating system name and version.

The GetWebServiceInfo method has the following signature:

[WebServiceInfo](#) GetWebServiceInfo();

The response consists of:

- The Web Service Name
- The Web Service Version
- The Native nCode Library Version
- The Address Data Effective Date
- The Name of the Web Server Application Running the Web Service
- The Name of the Virtual Machine Running under the Web Server Application
- The Name of the Operating System Running the Web Server Application
- The success status of the method call indicator (true or false)
- The message that describes the result of the method
- Name of the company the software is licensed to
- First address line of the company the software is licensed to
- Second address line of the company the software is licensed to
- Indicator whether the version data file match the license info
- Reason why the version data file does not match the license info
- Indicator whether PoCAD data context is available
- SOA effective date, when PoCAD data context is available

**Note:** The GetWebServiceInfo method was introduced with nCodeWS4 version 4.2. In the version 4.7, the SOA related elements were added to the WebServiceInfo structure.

# nCodeWS4 Structures Reference

## nCodeWS4 Structures

The structures used by nCodeWS4 are listed here. For each structure, its role is described and its elements are listed. For each structure element, its data type and its description are given. All nCodeWS4 structures belong to "http://novamg.com/nCodeWS4/" namespace.

You may notice that the structures representing requests for address analysis, address browsing, address parsing, address transformation and address capture all contain three elements that determine how nCode conducts the operation in question. The address layout element defines how big your address record is, how many logical lines it is subdivided into, how long each of these address lines are, and what address elements can be expected to be found on each of these lines. The analysis mode element defines how the address lines are parsed and how their content is matched against the postal database. The correction style element defines how the results of the operation are formatted, including the letter case style and the accent style of the response.

Address lines that correspond to the address layout in question are provided as arrays of LineContent structures. It should be mentioned that empty address lines do not have to be passed in the respective requests. If a LineContent structure is passed and its line name does not match the address layout in question, it will be ignored. The response will contain all address lines found in an address layout, even if their content is empty.

The nCodeWS4 method response structures contain two elements that describe the outcomes of the operations. The return code element (intRetCode) indicates success or failure of the method call, while the message (strMessage) provides a description of the outcome. If method call failed, the message would normally contain the error that caused the failure.

---

## LineContent Structure

LineContent structure represents an address line in a given address layout and it consists of the line name and the line content.

Element	Data Type	Description
strName	String	Address Line name
strContent	String	Analysis Line content

---

## CStyleChange Structure

CStyleChange structure represents address style changes that are the result of address correction and optimization.

CStyleChange structure is new to nCodeWS4.

Element	Data Type	Description
---------	-----------	-------------

Category	<a href="#">CStyleCategory</a>	Address style change category
ChangeType	<a href="#">CStyleChangeType</a>	Address style change type

---

## ParserElements Structure

ParserElements structure represents address elements as provided in the address input. It covers all types of addresses (street, street served by route, PO Box, route, GD). Address elements that do not belong to a specific type of address will be empty. For example, in a street type address, StationName, StationType, StationQualifier, POBoxKeyword, POBoxNumber, RouteKeyword, RouteNumber and GDKeyword elements will be empty. If an address element was missing in the input address, the corresponding structure element will be empty. When guessing modes are used in address analysis, the ParserElements structure may contain address elements that are only based on the address input, not necessarily matching it completely.

**Note.** With the nCode 12.1 release, elements StandardExtraInfo and NonStandardExtraInfo were added to the ParserElements structure.

Element	Data Type	Description
StreetName	String	Street name
StreetType	String	Street type
StreetDirection	String	Street direction
StreetNumber	String	Street number
SuiteKeyword	String	Suite keyword
SuiteNumber	String	Suite number
StationName	String	Station name
StationType	String	Station type
StationQualifier	String	Station qualifier
POBoxKeyword	String	PO Box keyword
POBoxNumber	String	PO Box number
RouteKeyword	String	Route keyword
RouteNumber	String	Route number
GDKeyword	String	GD keyword
Municipality	String	Municipality
Province	String	Province
PostalCode	String	Postal code
StandardExtraInfo	String	Standard Extra Information
NonStandardExtraInfo	String	Non-Standard Extra Information

---

# AddressElements Structure

AddressElements structure represents addresses found in the address database. It covers all types of addresses (street, street served by route, PO Box, route, GD). Address elements that do not belong to a specific type of address will be empty. When AddressElements structure is obtained through address analysis, it may additionally contain specific street, unit or PO Box numbers found in the input address. It may also contain standard and/or non-standard extra information detected in the input address. When AddressElements structure is obtained through address browsing or address searches, it only contains street, unit or PO Box number ranges found in the address database. The DataSource element value can be 1 for data source being CPC or 2 for data source being DMTI. The UnitNotFound element was added with nCode 12.2. The BuildingType code is 1 for residential buildings, 2 for business buildings and 3 for non-LVRA type postal codes. The BuildingType element was added with nCode 12.6.

The DataSource element is new to nCodeWS4.

Element	Data Type	Description
TypeOfAddress	<a href="#">AES_TypeOfAddress</a>	Type of the address
LVRType	<a href="#">AES_TypeOfLVR</a>	Type of the LVR
LVRName	String	The name of the LVR
BranchName	String	Government LVR names may have branches
LVRLanguage	<a href="#">AES_LVRLanguage</a>	The language LVR name was given in
StandardExtraInfo	String	Standard extra information
NonStandardExtraInfo	String	Non-standard extra information
StreetName	String	Street name
StreetType	String	Street type
StreetDirection	String	Street direction
StreetNumber	String	Street number
StreetNumberFrom	String	Street number from
StreetNumberTo	String	Street number to
SequenceCode	<a href="#">AES_SequenceCode</a>	Street number sequence code
StreetNumberSuffix	String	Street number suffix
StreetNumberSuffixFrom	String	Street number suffix from
StreetNumberSuffixTo	String	Street number suffix to
SuiteKeyword	String	Suite keyword
SuiteNumber	String	Suite number
SuiteNumberFrom	String	Suite number from
SuiteNumberTo	String	Suite number to
StationName	String	Station name

StationType	String	Station type
StationQualifier	String	Station qualifier
POBoxKeyword	String	PO Box keyword
POBoxNumber	String	PO Box number
POBoxNumberFrom	String	PO Box number from
POBoxNumberTo	String	PO Box number to
RouteKeyword	String	Route keyword
RouteNumber	String	Route number
GDKeyword	String	GD keyword
Municipality	String	Municipality
DirectoryArea	String	Directory area
Province	String	Province
PostalCode	String	Postal code
DataSource	Integer	Data source
UnitNotFound	Integer	Unit not found flag
BuildingType	Integer	The building type code

---

## AnalyzeInput Structure

AnalyzeInput structure represents the request parameter for the AnalyzeAddress method.

Element	Data Type	Description
strLayout	String	Address Layout name
strAnalysis	String	Analysis Mode name
strCorrection	String	Correction Style name
AnalysisType	<a href="#">NAA_AnalysisType</a>	Type of address analysis
udtLinesIn	<a href="#">LineContent[]</a>	List of input address lines

---

## AnalyzeOutput Structure

AnalyzeOutput structure represents response to the AnalyzeAddress method call. [The status of address analysis](#) is a string with the length of 3. The first position in the string contains the primary status of the analyzed address record, the second position contains the secondary status of the analyzed record and the third position contains urban status of the analyzed record.

The StyleChanges element is new to nCodeWS4.

Element	Data Type	Description
udtLinesOut	<a href="#">LineContent</a> []	List of output address lines
strStatus	String	Status of address analysis
strMessage	String	Description of the outcome of the method call
intAlternatives	Integer	Number of address alternatives found
intUnique	Integer	Unique identifier for the matching nCode database record
StyleChanges	<a href="#">CStyleChange</a> []	List of address style changes
HashCode	String	The hash code associated with the input address
Country	String	Foreign country name, if the address was declared foreign
intRetCode	Integer	Return value of the method call

---

## AnalyzeGetAlternativesInput Structure

AnalyzeGetAlternativesInput structure represents the request parameter for the AnalyzeGetAlternatives method. Address correction is implied with this request.

Element	Data Type	Description
strLayout	String	Address Layout name
strAnalysis	String	Analysis Mode name
strCorrection	String	Correction Style name
udtLinesIn	<a href="#">LineContent</a> []	List of input address lines

---

## ALTMessage Structure

ALTMessage structure is used to describe how an alternative address differs from an address record analyzed. A single such difference consists of message category, message detail, related address type, corrected address type value and the error message description.

Element	Data Type	Description
MessageCategory	<a href="#">ALT_MessageCategory</a>	Message category
MessageDetail	<a href="#">ALT_MessageDetail</a>	Message detail
AddressType	<a href="#">NAT_AddressType</a>	Address type related to the message
CorrectTypeValue	String	Corrected address type value, when applicable
TheMessage	String	Message description

---

## AlternativeAddress Structure

AlternativeAddress structure provides an alternative address for the address record analyzed. The Address part of the structure contains address elements that match nCode database. The Message part describes how the alternative address differs from the address record analyzed, through a list of error messages. Each error message contains message category, message detail, related address type, corrected address type value and the error message description.

Element	Data Type	Description
Address	<a href="#">AddressElements</a>	Alternative address
Message	<a href="#">ALTMessage[]</a>	List of error messages related to the alternative address

---

## AnalyzeGetAlternativesOutput Structure

AnalyzeGetAlternativesOutput structure represents response to the AnalyzeGetAlternatives method call.

[The status of address analysis](#) is a string with the length of 3. The first position in the string contains primary status of the analyzed address record, the second position contains the secondary status of the analyzed record and the third position contains urban status of the analyzed record.

The StyleChanges element is new to nCodeWS4.

Element	Data Type	Description
udtLinesOut	<a href="#">LineContent[]</a>	List of output address lines
intUnique	Integer	Unique identifier for the matching nCode database record
strStatus	String	Status of address analysis
intAlternatives	Integer	Number of address alternatives found
Alternatives	<a href="#">AlternativeAddress[]</a>	List of alternative addresses and associated error codes
strMessage	String	Description of the outcome of the method call
StyleChanges	<a href="#">CStyleChange[]</a>	List of address style changes
HashCode	String	The hash code associated with the input address
Country	String	Foreign country name, if the address was declared foreign
intRetCode	Integer	Return value of the method call

---

## FormatAddressInput Structure

FormatAddressInput structure represents the request parameter for the FormatAddress method. The Address element corresponds to a postal database record and it may contain street, unit or PO Box number ranges. If the Address element does not contain any ranges, the Number element can be uniquely determined from the record and therefore an empty Number value would be acceptable. In case of a street, unit or PO Box number range, a specific Number value from the range must be provided.

Element	Data Type	Description
strLayout	String	Address Layout name



strAnalysis	String	Analysis Mode name
strCorrection	String	Correction Style name
Address	<a href="#">AddressElements</a>	Address elements
Number	String	Specific number from street, unit or PO Box range

---

## FormatAddressOutput Structure

FormatAddressOutput structure represents response to the FormatAddress method call. If successful, the output address lines will be optimized, according to the given correction style.

Element	Data Type	Description
udtLinesOut	<a href="#">LineContent</a> []	List of output address lines
strMessage	String	Description of the outcome of the method call
intRetCode	Integer	Return value of the method call

---

## FormatPEInput Structure

FormatPEInput structure represents the request parameter for the FormatPE method. The address elements found in udtParserElements member may or may not represent a complete address. For example, parser elements obtained through street name database searches would not contain specific street numbers. Even so, such incomplete addresses can still be passed for formatting in the given address layout.

Element	Data Type	Description
strLayout	String	Address Layout name
strAnalysis	String	Analysis Mode name
strCorrection	String	Correction Style name
udtParserElements	<a href="#">ParserElements</a>	Parser elements to be formatted in the given address layout

---

## FormatPEOutput Structure

FormatPEOutput structure represents response to the FormatPE method call. Since the parser elements passed may not have represented a complete address, the output address lines will not be optimized.

Element	Data Type	Description
udtLinesOut	<a href="#">LineContent</a> []	List of output address lines
strMessage	String	Description of the outcome of the method call
intRetCode	Integer	Return value of the method call

---

## ParseAddressInput Structure

ParseAddressInput structure represents the request parameter for the ParseAddress method. Guessing mode flags tell the parser engine how to go about finding best matches for the address input provided by the caller.

The GuessingFlags element is new to nCodeWS4.

Element	Data Type	Description
strLayout	String	Address Layout name
strAnalysis	String	Analysis Mode name
strCorrection	String	Correction Style name
GuessingFlags	<a href="#">GuessingModes</a>	Guessing mode flags
udtLinesIn	<a href="#">LineContent</a> []	List of input address lines

---

## ParseAddressOutput Structure

ParseAddressOutput structure represents response to the ParseAddress method call. Every parser variant in the list of parser variants represents an alternative best interpretation of the address input provided. During the address parsing stage, address elements found in the input address only need to exist in the address database; they do not necessarily need to be related on the database record level in order for a parser variant to be created.

Element	Data Type	Description
PVariant	<a href="#">ParserElements</a> []	List of parser variants obtained through address parsing
strMessage	String	Description of the outcome of the method call
intRetCode	Integer	Return value of the method call
PVAddressesFound	Integer[]	List of addresses found counters, matching the PVariant list

---

## BrowseAddressExInput Structure

BrowseAddressExInput structure represents the request parameter for the BrowseAddressEx method. Guessing mode flags tell the parser engine how to go about finding best matches for the address input provided by the caller. The GuessingFlags element is new to nCodeWS4.

Element	Data Type	Description
strLayout	String	Address Layout name
strAnalysis	String	Analysis Mode name
strCorrection	String	Correction Style name
GuessingFlags	<a href="#">GuessingModes</a>	Guessing mode flags
udtLinesIn	<a href="#">LineContent</a> []	List of input address lines
MaxNamesToAddresses	Integer	If the number of names found is less than or equal to MaxNamesToAddresses, the resulting name list is converted to address list.

If the value of 0 is passed, the default value of 2 will be used. If a value greater than 1000 is passed, the value of 1000 will be used.

---

## BrowseAddressExOutput Structure

BrowseAddressExOutput structure represents response to the BrowseAddressEx method call. Every parser variant in the PVariant list of parser variants represents an alternative best interpretation of the address input provided. During the address browsing stage, address elements found in the input address not only need to exist in the address database, they also need to be related on the database record level. The ResultType selector defines whether the list of addresses or the list of names is the result of address browsing. In case of inconsistent address input, the result of address browsing could be that both address and name lists are empty. The list of indexes PVIndex relates the address list elements or the name list elements to specific parser variants in the PVariant list, telling us which address or name was found based on which parser variant.

Element	Data Type	Description
PVariant	<a href="#">ParserElements[]</a>	List of parser variants obtained through address browsing
ResultType	<a href="#">DAB_ResultType</a>	Result type defines whether address list or name list applies
Address	<a href="#">AddressElements[]</a>	List of addresses obtained through address browsing
Name	<a href="#">ParserElements[]</a>	List of names obtained through address browsing
PVIndex	Integer[]	List of indexes relates Address and Name lists to PVariant list
strMessage	String	Description of the outcome of the method call
intRetCode	Integer	Return value of the method call

---

## TransformInput Structure

TransformInput structure represents the request parameter for the TransformAddress method. The list of input address lines relates to the first address layout. Addresses will be transformed from the first address layout into the second address layout.

Element	Data Type	Description
strFirstLayout	String	First Address Layout name
strSecondLayout	String	Second Address Layout name
strAnalysis	String	Analysis Mode name
strCorrection	String	Correction Style name
udtLinesIn	<a href="#">LineContent[]</a>	List of input address lines

---

## TransformOutput Structure

TransformOutput structure represents response to the TransformAddress method call. If the input address was valid or correctable, the transformed address will be a valid optimized address. In case of a non-correctable input address with multiple alternatives found, only the common address elements among all alternatives will be present in the transformed address.

Element	Data Type	Description
udtLinesOut	<a href="#">LineContent[]</a>	List of output address lines
strMessage	String	Description of the outcome of the method call
intRetCode	Integer	Return value of the method call

---

## BrowsingRequest Structure

BrowsingRequest structure represents a set of conditions used in databases searches. The structure is used as the request parameter for the SearchForAddresses and SearchForNames methods. Empty value for NumberFrom, NumberTo, Name, Type, Direction, Municipality, Province or PostalCode elements means that the corresponding condition does not exist, so any value will be allowed in its place. If IncludeAll value is on, it overrides other specific include conditions, so that all street, street served by route, PO Box, route, GD, urban and rural addresses will be included in the results of the search. If IncludeAll value is off, the combination of specific include conditions will determine the results of the search. In order to gain explicit control over the search and formatting of the results of the search, nCodeWS4 has added the following elements: ApplyMisspellings, ApplyMissingWords, AccentedInput, AccentedOutput and LowercaseOutput. Corresponding nCodeWS3 behaviours were implicit and depended on the selected analysis mode and correction style. RouteKeyword and RouteNumber elements were added with nCode 12.1. The elements ApplyAltMnc, UseMncAsGda and MaxItems were added with nCode 12.2.

Element	Data Type	Description
NumberFrom	String	Street, PO Box or unit number from condition
NumberTo	String	Street, PO Box or unit number to condition
AllInRange	<a href="#">NB_AllInRange</a>	Street, PO Box or unit number range inclusion condition
Name	String	Street name or station name condition
Type	String	Street type or station type condition
Direction	String	Street direction or station qualifier condition
Municipality	String	Municipality condition
Province	String	Province condition
PostalCode	String	Postal code condition
IncludeAll	<a href="#">NB_IsIncluded</a>	Should results include all address types
IncludeStreet	<a href="#">NB_IsIncluded</a>	Should results include street addresses
IncludeSSBR	<a href="#">NB_IsIncluded</a>	Should results include street served by route addresses
IncludePOBox	<a href="#">NB_IsIncluded</a>	Should results include station PO Box addresses
IncludeRoute	<a href="#">NB_IsIncluded</a>	Should results include station route addresses
IncludeGD	<a href="#">NB_IsIncluded</a>	Should results include station GD addresses
IncludeUrban	<a href="#">NB_IsIncluded</a>	Should results include urban addresses

IncludeRural	<a href="#">NB_IsIncluded</a>	Should results include rural addresses
TargetType	<a href="#">NB_TargetType</a>	The type of search to be performed
ApplyMisspellings	Boolean	Should results include misspelled street and station names
ApplyMissingWords	Boolean	Should results include street and station names missing words
AccentedInput	Boolean	Should accented input be expected
AccentedOutput	Boolean	Should accented output be produced
LowercaseOutput	Boolean	Should lowercase output be produced
RouteKeyword	String	Route Keyword Condition
RouteNumber	String	Route Number Condition
ApplyAltMnc	Boolean	Should alternate municipality names be used in searches
UseMncAsGda	Boolean	Should municipality be used as directory area in searches
MaxItems	Integer	The maximum number of items to be returned

---

## NameElements Structure

NameElements structure is used to express results of name searches against the database. When search type is search for street or station names, Name, Type, Direction, Municipality, Directory Area and Province elements will be defined, while FSA element will be empty. When search type is search for municipalities, only Municipality, Directory Area and Province elements will be defined, while Name, Type, Direction and FSA elements will be empty. When search type is search for FSA-s, only FSA and Province elements will be defined, while Name, Type, Direction, Municipality and Directory Area elements will be empty. When search type is search for postal codes, only FSA (as postal code), Municipality, Directory Area and Province elements will be defined, while Name, Type and Direction elements will be empty. The DirectoryArea element was added with nCode 12.2.

Element	Data Type	Description
Name	String	Street name or station name
Type	String	Street type or station type
Direction	String	Street direction or station qualifier
Municipality	String	Municipality
Province	String	Province
FSA	String	Forward Sortation Area
DirectoryArea	String	Greater Directory Area Name

---

## AddressesFound Structure

AddressesFound structure represents response to the SearchForAddresses method call. In case of inconsistent search conditions, the resulting list of addresses will be empty. The ActualAddressesFound element was added with nCode 12.2.

Element	Data Type	Description
Address	<a href="#">AddressElements</a> []	List of addresses found
intRetCode	Integer	Return value of the method call
strMessage	String	Description of the outcome of the method call
ActualAddressesFound	Integer	Actual number of items found by native nCode

---

## NamesFound Structure

NamesFound structure represents response to the SearchForNames method call. In case of inconsistent search conditions, the resulting list of names will be empty. The ActualNamesFound element was added with nCode 12.2.

Element	Data Type	Description
Name	<a href="#">NameElements</a> []	List of names found
intRetCode	Integer	Return value of the method call
strMessage	String	Description of the outcome of the method call
ActualNamesFound	Integer	Actual number of items found by native nCode

---

## NAMAnalysisModeGen Structure

NAMAnalysisModeGen structure describes general, non address element type specific, analysis mode settings. MissingWordsMode and MissingRouteInfoMode elements are new to nCodeWS4. The AcceptUnit element was added with nCode 12.2. The LanguageMode element was added with nCode 12.6. The Owner element was added with nCode 12.6.

Element	Data Type	Description
AModeName	String	Analysis mode name
UseAlteranteWords	<a href="#">NAM_PropertyFlag</a>	Use alternate words in analysis setting
InputIsAccented	<a href="#">NAM_PropertyFlag</a>	Input is accented setting
MaxChangedKeyElements	<a href="#">NAM_ChangedKeyElements</a>	Maximum number of changed key address elements setting
SpellingLevel	<a href="#">NAM_SpellingLevel</a>	Spelling level setting
MissingWordsMode	<a href="#">NAM_MissingWordsMode</a>	Missing words mode setting
MissingRouteInfoMode	<a href="#">NAM_MissingRouteInfoMode</a>	Missing route information mode setting
AcceptUnit	<a href="#">NAM_AcceptUnit</a>	Accept unit info not found in PCAD setting

LanguageMode	<a href="#">NAM_LanguageMode</a>	The language mode of address analysis.
Owner	String	The owner of the analysis mode.

---

## NAMAnalysisModeType Structure

NAMAnalysisModeType structure describes address element type specific analysis mode settings.

Element	Data Type	Description
AddressType	<a href="#">NAT_AddressType</a>	Address element type
SeparatedByBlanks	<a href="#">NAM_PropertyFlag</a>	Is address element type separated by blanks
KeyAddressElement	<a href="#">NAM_PropertyFlag</a>	Is address element type considered a key address element
CanBeChanged	<a href="#">NAM_PropertyFlag</a>	Can address element type be changed in a corrected address

---

## AnalysisModeOutput Structure

AnalysisModeOutput structure represents response to the GetAnalysisMode method call. Some analysis modes may not be SOA compliant, meaning that they cannot be used to generate statements of accuracy (SOA). Depending on the general analysis mode settings, each analysis mode implies default guessing mode flag values, which would be used in address browsing, address parsing and address capture, if specific guessing mode flags were not provided explicitly.

SOACompliant, SOAReason and DefaultGuessingFlags elements are new to nCodeWS4.

Element	Data Type	Description
AMGeneralInfo	<a href="#">NAMAnalysisModeGen</a>	General analysis mode settings
AMTypeInfo	<a href="#">NAMAnalysisModeType</a> []	List of address element type specific analysis mode settings
SOACompliant	<a href="#">NAM_SOACompliant</a>	Is the analysis mode SOA compliant
SOAReason	String	If the analysis mode is not SOA compliant, provides the reason why
DefaultGuessingFlags	<a href="#">GuessingModes</a>	Default guessing mode flags
intRetCode	Integer	Return value of the method call
strMessage	String	Description of the outcome of the method call

---

## AnalysisModeInput Structure

AnalysisModeInput structure represents the request parameter for the SetAnalysisMode method.

Element	Data Type	Description
AMGeneralInfo	<a href="#">NAMAnalysisModeGen</a>	General analysis mode settings
AMTypeInfo	<a href="#">NAMAnalysisModeType</a> []	List of address element type specific analysis mode settings

---

## OperationResult Structure

OperationResult structure represents response to the SetAddressLayout, SetAnalysisMode and SetCorrectionStyle method calls.

Element	Data Type	Description
intRetCode	Integer	Return value of the method call
strMessage	String	Description of the outcome of the method call

---

## CorrectionStyleOutput Structure

CorrectionStyleOutput structure represents response to the GetCorrectionStyle method call. Some correction styles may not be SOA compliant, meaning that they cannot be used to generate statements of accuracy (SOA).

SOACompliant and SOAReason elements are new to nCodeWS4.

Element	Data Type	Description
CorrectionStyle	<a href="#">NCSCorrectionStyle</a>	Correction style settings
SOACompliant	<a href="#">NCS_SOACompliant</a>	Is the correction style SOA compliant
SOAReason	String	If the correction style is not SOA compliant, provides the reason why
intRetCode	Integer	Return value of the method call
strMessage	String	Description of the outcome of the method call

---

## NCSCorrectionStyle Structure

NCSCorrectionStyle structure describes general correction style settings.

StreetNameOptimum, MunicipalityNameOptimum, and PushLines elements are new to nCodeWS4. The Owner element was added with nCode 12.6.

Element	Data Type	Description
CStyleName	String	Correction style name
Lettercase	<a href="#">NCS_Lettercase</a>	Letter case style
Accent	<a href="#">NCS_Accent</a>	Accent style
UnitForm	<a href="#">NCS_UnitForm</a>	Unit information format style
Punctuation	<a href="#">NCS_Punctuation</a>	Punctuation style
PCForm	<a href="#">NCS_PCForm</a>	Postal code format style
ExtraInfo	<a href="#">NCS_ExtraInfo</a>	Extra information style
StrTypeKWOptimum	<a href="#">NCS_KWOptimum</a>	Street type optimum style
StrTypeMSOptimum	<a href="#">NCS_MSOptimum</a>	Street type misspelling style



StrDirKWOptimum	<a href="#">NCS_KWOptimum</a>	Street direction optimum style
StrDirMSOptimum	<a href="#">NCS_MSOptimum</a>	Street direction misspelling style
UnitKWOptimum	<a href="#">NCS_KWOptimum</a>	Unit keyword optimum style
UnitMSOptimum	<a href="#">NCS_MSOptimum</a>	Unit keyword misspelling style
StnTypeKWOptimum	<a href="#">NCS_KWOptimum</a>	Station type optimum style
StnTypeMSOptimum	<a href="#">NCS_MSOptimum</a>	Station type misspelling style
BoxKWOptimum	<a href="#">NCS_KWOptimum</a>	PO Box keyword optimum style
BoxMSOptimum	<a href="#">NCS_MSOptimum</a>	PO Box keyword misspelling style
RouteKWOptimum	<a href="#">NCS_KWOptimum</a>	Route keyword optimum style
RouteMSOptimum	<a href="#">NCS_MSOptimum</a>	Route keyword misspelling style
GDKWOptimum	<a href="#">NCS_KWOptimum</a>	GD keyword optimum style
GDMSOptimum	<a href="#">NCS_MSOptimum</a>	GD keyword misspelling style
ProvKWOptimum	<a href="#">NCS_KWOptimum</a>	Province optimum style
ProvMSOptimum	<a href="#">NCS_MSOptimum</a>	Province misspelling style
StreetNameOptimum	<a href="#">NCS_StreetNameOptimum</a>	Street name optimization style
MunicipalityNameOptimum	<a href="#">NCS_MunicipalityNameOptimum</a>	Municipality name optimization style
PushLines	<a href="#">NCS_PushLines</a>	Push lines style
Owner	String	The owner of the correction style.

---

## KWDescription Structure

KWDescription structure defines a single keyword. Not all keywords would be present in the CPC address database, which is indicated by the CPCPresent element. For example, alternate keyword values would normally not be found in the address database, since the address database contains only optimum keyword values.

Element	Data Type	Description
Keyword	String	Keyword value
CPCPresent	<a href="#">KW_CPCDBPresent</a>	Is the keyword value present in address database

---

## KWRequest Structure

KWRequest structure represents the request parameter for the GetKeywords method. It defines three filters for keyword searches: the address type of the keyword, the language of the keyword and whether only optimum values should be considered in the keyword search.

Element	Data Type	Description
LangInd	<a href="#">KW_Language</a>	Keyword language
OptOnlyInd	<a href="#">KW_OptimumOnly</a>	Optimum only indicator

---

## GetKeywordsOutput Structure

GetKeywordsOutput structure represents response to the GetKeywords method call.

Element	Data Type	Description
Keyword	<a href="#">KWDescription</a> []	List of keywords found
strMessage	String	Description of the outcome of the method call
intRetCode	Integer	Return value of the method call

---

## AddressLine Structure

AddressLine structure represents address lines in a given address layout, including line name, line start, line length and the list of address types that can be found in the address line. Address line start can be anywhere between one and the address record length.

Element	Data Type	Description
strLineName	String	Address line name
intLineStart	Integer	Address line start
intLineLength	Integer	Address line length
Types	<a href="#">NAT_AddressType</a> []	List of address types found in the address line

---

## AddressLayout Structure

AddressLayout structure represents response to the GetLayoutDetails method call. It contains all of the elements that define an address layout. The Owner element was added with nCode 12.6.

Element	Data Type	Description
strLayoutName	String	Address layout name
intRecordLength	Integer	Address record length
intTotalLines	Integer	Total number of address lines in the address layout
Lines	<a href="#">AddressLine</a> []	List of address lines in the address layout
intRetCode	Integer	Return value of the method call
strMessage	String	Description of the outcome of the method call
Owner	String	The owner of the address layout.

---

## NewAddressLayout Structure

NewAddressLayout structure represents the request parameter for the SetAddressLayout method. It contains all of the elements that define an address layout. The Owner element was added with nCode 12.6.

Element	Data Type	Description
strLayoutName	String	Address layout name
intRecordLength	Integer	Address record length
intTotalLines	Integer	Total number of address lines in the address layout
Lines	<a href="#">AddressLine[]</a>	List of address lines in the address layout
Owner	String	The owner of the address layout.

---

## ParamList Structure

ParamList structure represents response to the GetParamList method call. Depending on the request type, it can contain a list of address layouts, analysis modes or correction styles.

Element	Data Type	Description
strParamName	String[]	List of address layouts, analysis modes or correction styles
intRetCode	Integer	Return value of the method call
strMessage	String	Description of the outcome of the method call

---

## RemoveParamRequest Structure

RemoveParamRequest structure represents the request parameter for the RemoveParameter method.

Element	Data Type	Description
Type	<a href="#">NPAR_ParameterType</a>	Specifies whether the parameter name is address layout, analysis mode or correction style
Parameter	String	The name of address layout, analysis mode or correction style to be removed from the currently defined lists

---

## GetParamRequest Structure

GetParamRequest structure represents the request parameter for the GetParamListEx method.

Element	Data Type	Description
Type	<a href="#">NPAR_ParameterType</a>	Specifies whether the parameter name is address layout, analysis mode or correction style
Owner	String	The name of the owner of the address layout, analysis mode or the correction style.

---

## AltMncRecord Structure

AltMncRecord structure serves a dual purpose in alternate municipality searches. It can provide both conditions on alternate municipalities and represent alternate municipalities found. The FSACondition element is used to further narrow validity of alternate municipalities to only those addresses that belong to the given forward sortation area (FSA). Province condition states that the relationship between the municipality name and the alternate municipality name is restricted to the given province.

AltMncRecord structure is new to nCodeWS4.

Element	Data Type	Description
MunicipalityName	String	Municipality name
AltMunicipalityName	String	Alternate municipality name
TypeOfAlternative	<a href="#">AltMncType</a>	Type of alternate municipality name
ValidityIndicator	<a href="#">AltMncValidityIndicator</a>	Alternate municipality name validity indicator
FSACondition	String	FSA condition
ProvinceCondition	String	Province condition

---

## AltMncSearchInput Structure

AltMncSearchInput structure represents the request parameter for the SearchForAltMunicipalities method. Empty condition values means that any values in their place are acceptable.

AltMncSearchInput structure is new to nCodeWS4.

Element	Data Type	Description
SearchConditions	<a href="#">AltMncRecord</a>	Alternate municipality search conditions

---

## AltMncSearchOutput Structure

AltMncSearchOutput structure represents response to the SearchForAltMunicipalities method call. AltMncSearchOutput structure is new to nCodeWS4.

Element	Data Type	Description
SearchResult	<a href="#">AltMncSearchResult</a>	Return value of the method call
AltMncRecords	<a href="#">AltMncRecord</a> []	List of alternate municipalities found
Message	String	Description of the outcome of the method call

---

## AltStrRecord Structure

AltStrRecord structure serves a dual purpose in alternate street searches. It can provide both conditions on alternate streets and represent alternate streets found. Municipality name and province define the conditions under which alternate street name, alternate street type and alternate street direction represent a valid alternative to the street name, street type and street direction.

AltStrRecord structure is new to nCodeWS4.

Element	Data Type	Description
StreetName	String	Street name
StreetType	String	Street type
StreetDirection	String	Street direction
AltStreetName	String	Alternate street name
AltStreetType	String	Alternate street type
AltStreetDirection	String	Alternate street direction
MunicipalityCondition	String	Municipality name condition
ProvinceCondition	String	Province condition

---

## AltStrSearchInput Structure

AltStrSearchInput structure represents the request parameter for the SearchForAltStreets method. Empty condition values means that any values in their place are acceptable. Street names are sometimes missing their street types and/or street directions. Therefore, when providing alternate street search conditions, we need to specify whether empty street type and empty street direction are explicitly empty or any value in their place is allowed.

AltStrSearchInput structure is new to nCodeWS4.

Element	Data Type	Description
SearchConditions	<a href="#">AltStrRecord</a>	Alternate street search conditions
IsEmptyStreetTypeExplicit	Boolean	Is empty street type explicit flag
IsEmptyStreetDirectionExplicit	Boolean	Is empty street direction explicit flag
IsEmptyAltStreetTypeExplicit	Boolean	Is empty alternate street type explicit flag
IsEmptyAltStreetDirectionExplicit	Boolean	Is empty alternate street direction explicit flag

---

## AltStrSearchOutput Structure

AltStrSearchOutput structure represents response to the SearchForAltStreets method call. AltStrSearchOutput structure is new to nCodeWS4.

Element	Data Type	Description
SearchResult	<a href="#">AltStrSearchResult</a>	Return value of the method call
AltStrRecords	<a href="#">AltStrRecord</a> []	List of alternate streets found
Message	String	Description of the outcome of the method call

---

## GuessingModes Structure

GuessingModes structure is used to instruct the nCode parser engine on how to go about finding best matches for the address input provided. Normally, nCode parser engine recognizes only street, station and municipalities names that are

spelled in full. However, in certain situations, it is preferable for nCode to engage in speculation about what the input provided could have meant. For example, if a misspelled name or a name with missing words appeared in the input, nCode would not be able to directly match it against the database, unless it was instructed to do so via the guessing mode flags. On the downside, using guessing modes results in somewhat slower address parsing and address analysis. It can also result in a much larger number of matches than the searches with all of the guessing modes turned off. Guessing of street, station and municipality names that start with a given prefix is very useful in address capture scenarios, since nCode may be able to narrow down the choices with only a small subset of the full address typed in. If the UseDefaults element is set to true, the guessing mode flag values will be implicitly determined by the currently used analysis mode. When explicit control of the guessing mode flags is needed, set the UseDefaults element to false.

Analysis mode settings implicitly define the default values of the guessing mode flags, using the following rules:

- If the analysis mode spelling level is less than NAM\_SPELLING\_LEVEL\_FIVE, guessing of misspelled words in street/station names is on, otherwise, it is off.
- If the analysis mode setting for guessing missing words in address analysis is not NAM\_GUESS\_MW\_NEVER, guessing of missing words in street/station names is on, otherwise, it is off.
- If both guessing of misspelled words in street/station names and guessing of missing words in street/station names is on, guessing of street/station names that start with a given prefix is on, otherwise, it is off.
- If both guessing of misspelled words in street/station names and guessing of missing words in street/station names is on, guessing of municipality names that start with a given prefix is on, otherwise, it is off.
- If any of street/station name guessing mode flags is on, preferring street name to municipality name is on, otherwise, it is off.

GuessingModes structure is new to nCodeWS4.

Element	Data Type	Description
UseDefaults	Boolean	Should guessing defaults be used
GuessSNMisspellings	Boolean	Is guessing of misspelled words in street/station names on
GuessSNMissingWords	Boolean	Is guessing of missing words in street/station names on
GuessSNStartsWith	Boolean	Is guessing of street/station names that start with a given prefix on
GuessMNStartsWith	Boolean	Is guessing of municipality names that start with a given prefix on
PreferStreetToMnc	Boolean	Is preferring street names to municipality names, when following a number, on
IgnoreAltMncInSearches	Boolean	Is ignoring alternate municipality names in searches on

---

## OneStopAnalyzeInput Structure

OneStopAnalyzeInput structure represents the request parameter for the OneStopAnalyze method. The selected alternative element must be passed exactly as originally provided, otherwise, the parsing of the address alternative may fail. If no alternative was selected, the selected alternative element should be empty. Guessing mode flags tell the parser engine how to go about finding best matches for the address input provided by the caller. Maximum number of alternatives element is there to limit the number of alternatives provided in the response to the OneStopAnalyze method call. If the value passed in the MaxAlternatives element is zero, the number of alternatives provided in the response is unlimited.

OneStopAnalyzeInput structure is new to nCodeWS4.

Element	Data Type	Description
AddressLayout	String	Address layout name

AnalysisMode	String	Analysis mode name
CorrectionStyle	String	Correction style name
GuessingFlags	<a href="#">GuessingModes</a>	Guessing mode flags
AddressLinesIn	<a href="#">LineContent[]</a>	List of input address lines
SelectedAlternative	String	Selected alternative
MaxAlternatives	Integer	Maximum number of alternatives in the response
EnableAltDetails	Boolean	Flag indicating whether address elements can be retrieved for all alternatives. The default value of false means that address elements can be retrieved for complete and suggested addresses only.
MaxNamesToAddresses	Integer	If the number of names found is less than or equal to MaxNamesToAddresses, the resulting name list is converted to address list. If the value of 0 is passed, the default value of 2 will be used. If a value greater than 1000 is passed, the value of 1000 will be used.

**Note.** With the nCode 12.1 release, element EnableAltDetails was added to the OneStopAnalyzeInput structure. Since OneStopAnalyzeInput structure is passed to nCodeWS4, the EnableAltDetails element is defined as optional. If missing in the request, it defaults to false, implying the previous behavior and thus preserving backwards compatibility with older nCodeWS4 client applications. Using the EnableAltDetails default value is also recommended for performance reasons.

---

## OneStopAnalyzeOutput Structure

OneStopAnalyzeOutput structure represents response to the OneStopAnalyze method call. If the SuccessStatus element value is false, the Message element will contain the related error message and other structure elements will not be relevant. If the AddressComplete element value is true, the AddressLinesOut element will provide a fully formatted address and the AddressDetails element will provide its address elements. The Alternatives element provides a list of alternatives of AlternativeListType type (addresses, names or empty list). The ActualAlternativesFound element tells us how many alternatives were actually found during the OneStopAnalyze method call. If this number exceeded the maximum number of alternatives passed in the request, not all of the alternatives found would be present in the Alternatives list. When the oneStop analysis narrows down the choices to a single address record containing ranges, the AddressComplete value will be false and the SuggestedLines element will contain a formatted address with a number placeholder (?###) in place of a missing number. The DataSource element value can be 1 for data source being CPC, 2 for data source being DMTI and 0 if data source information is not applicable.

OneStopAnalyzeOutputstructure is new to nCodeWS4.

Element	Data Type	Description
SuccessStatus	Boolean	Success status of the method call
AddressComplete	Boolean	Address complete indicator
AddressLinesOut	<a href="#">LineContent[]</a>	List of output address lines
SuggestedLines	<a href="#">LineContent[]</a>	List of suggested address lines
AddressDetails	<a href="#">AddressElements[]</a>	Address element details
AlternativeListType	<a href="#">AltListType</a>	Type of the list of alternatives
ActualAlternativesFound	Integer	Actual alternatives found
Alternatives	String[]	The list of alternatives

Message	String	Description of the outcome of the method call
DataSource	Integer	Data source

---

## WebServiceInfo Structure

WebServiceInfo structure represents response to the GetWebServiceInfo method call. If the SuccessStatus element value is false, the Message element will contain the related error message and other structure elements will not be relevant. WebServerInfo element represents the name of the Web Server application running the Web Service. Its values can be WCF, WebLogic, JBoss, Tomcat or WebSphere, coupled with their version number. For example, JBoss Web 7.0.13.Final. VMInfo element represents the respective virtual machine name and its version. For example, Oracle Java 1.8.0\_144 64-bit. OSInfo represents the operating system running the Web Service. For example, Windows 10 version 10.0. WebServiceVersion represents a combination of the WSDL version and its implementation number. It comes in the form major.minor.build. For nCodeWS4, the major version is fixed to 4. The minor version describes the WSDL change version and the build number represents implementation changes within major.minor version.

WebServiceInfo structure is new to nCodeWS4 version 4.2. In the version 4.7, the SOA related elements were added to the WebServiceInfo structure: Company, Address1, Address2, VersionFileMatch, VersionFileReason, SOAEnabled, SOAEffected.

Element	Data Type	Description
WebServiceName	String	Web Service Name
WebServiceVersion	String	Web Service Version
NativeLibraryVersion	String	Native nCode Library Version
DataVersion	String	Address Data Effective Date
WebServerInfo	String	Web Server Application Running the Web Service
VMInfo	String	Virtual Machine Running under the Web Server
OSInfo	String	Operating System Running the Web Server
SuccessStatus	Boolean	Success status of the method call
Message	String	Description of the outcome of the method call
Company	String	Name of the company the software is licensed to
Address1	String	First address line of the company the software is licensed to
Address2	String	Second address line of the company the software is licensed to
VersionFileMatch	Boolean	Does the version data file match the license info
VersionFileReason	String	Reason why the version data file does not match the license info
SOAEnabled	Boolean	SOAEnabled is true if and only if PoCAD data context is available
SOAEffected	String	If SOA is enabled, provides the SOA effective date



---

## AnalyzeRecordsInput

AnalyzeRecordsInput structure represents the request parameter for the AnalyzeRecords method.

Element	Data Type	Description
AddressLayout	String	Address Layout name
AnalysisMode	String	Analysis Mode name
CorrectionStyle	String	Correction Style name
AnalysisType	<a href="#">NAA_AnalysisType</a>	Type of address analysis
RecordsIn	String[]	List of input address records
CreateAltMessages	Boolean	Should alternate messages be created

---

## ResultingRecord

ResultingRecord structure represents the results of processing an individual input address record, given in the AnalyzeRecordsInput structure.

Element	Data Type	Description
Record	String	The resulting address record
Status	String	Status of address analysis
Message	String	Description of the outcome of the method call
Alternatives	Integer	Number of address alternatives found
AltMessages	<a href="#">ALTMessage[]</a>	List of messages related to the first alternative address

---

## AnalyzeRecordsOutput

AnalyzeRecordsOutput structure represents response to the AnalyzeRecords method call.

Element	Data Type	Description
RecordsOut	<a href="#">ResultingRecord[]</a>	List of output address records and their analysis details
DataContext	Boolean	Which data context (PCAD=1 or PoCAD=2) was used
Result	String	Success status of the method call
Message	String	Description of the outcome of the method call



# nCodeWS4 Enumerations Reference

## nCodeWS4 Enumerations

The enumerations used by nCodeWS4 are listed here. For each enumeration, its description is provided. All nCodeWS4 enumerations belong to "http://novamg.com/nCodeWS4/" namespace.

Using enumerations makes working with nCodeWS4 easier in languages (such as ASP.NET) with helper functions similar to Visual Studio's IntelliSense feature.

---

## AES\_LVRLanguage Enumeration

LVR Language Type enumeration represents language types of LVR names.

Enumeration	Description
AES_EnglishLVR	English Name LVR
AES_FrenchLVR	French Name LVR

---

## AES\_SequenceCode Enumeration

Street Number Sequence enumeration represents types of street number ranges.

Enumeration	Description
AES_OddNumbers	Odd Numbers Only
AES_EvenNumbers	Even Numbers Only
AES_ConsecutiveNumbers	Consecutive Numbers
AES_SSBRBoxNumbers	Street Served By Route Box Numbers (Alphanumeric)

---

## AES\_TypeOfAddress Enumeration

Type of Address enumeration represents types of Canadian addresses, as defined by CPC.

Enumeration	Description
-------------	-------------

AES_StreetAddress	Street Address
AES_SSBRAddress	Street Served By Route Address
AES_POBoxAddress	PO Box Address
AES_RouteAddress	Rural Route Address
AES_GDAddress	General Delivery Address

---

## AES\_TypeOfLVR Enumeration

LVR Type enumeration represents types of LVR-s, as defined by CPC.

Enumeration	Description
AES_Building	Building
AES_StreetLVR	Street Address LVR
AES_GovStreetLVR	Street Address Government LVR
AES_POBoxLVR	PO Box Address LVR
AES_GovPOBoxLVR	PO Box Address Govern. LVR
AES_GDLVR	General Deliver LVR
AES_NotAnLVR	Not An LVR Address

---

## AltListType Enumeration

List Of Alternatives Type enumeration is used in oneStop address capture to define the type of the list of alternatives. AltListType enumeration is new to nCodeWS4.

Enumeration	Description
Empty	List Of Alternatives Is Empty
Addresses	List Of Alternatives Represents Addresses
Names	List Of Alternatives Represents Names

---

## AltMncSearchResult Enumeration

Alternate Municipality Search Result enumeration represents possible outcomes of alternate municipality searches. AltMncSearchResult enumeration is new to nCodeWS4.

Enumeration	Description
Successful	Search Was Successful
InternalError	Search Resulted In Internal nCode Error

InvalidMncCondition	Invalid Municipality Name Search Condition Was Provided
InvalidAltMncCondition	Invalid Alt. Municipality Name Search Condition Was Provided
InvalidFSACondition	Invalid FSA Search Condition Was Provided
InvalidProvinceCondition	Invalid Province Search Condition Was Provided

---

## AltMncType Enumeration

Alternate Municipality Type enumeration represents types of alternate municipality names. AltMncType enumeration is new to nCodeWS4.

Enumeration	Description
Valid	Valid Alternate Municipality Name
Invalid	Invalid Alternate Municipality Name
Abbreviation13	13 Character Abbreviation Of The Alternate Municipality Name
Abbreviation18	18 Character Abbreviation Of The Alternate Municipality Name
AnyType	Any Alternate Municipality Name Type

---

## AltMncValidityIndicator Enumeration

Alternate Municipality Validity Indicator enumeration indicates acceptability of alternate municipality names. AltMncValidityIndicator enumeration is new to nCodeWS4.

Enumeration	Description
Acceptable	Alternate Municipality Name Is Acceptable
Unacceptable	Alternate Municipality Name Is Not Acceptable
AnyValidityIndicator	Any Alternate Municipality Name Validity Indicator

---

## AltStrSearchResult Enumeration

Alternate Street Search Result enumeration represents possible outcomes of alternate street searches. AltStrSearchResult enumeration is new to nCodeWS4.

Enumeration	Description
Successful	Search Was Successful
InternalError	Search Resulted In Internal nCode Error
InvalidStretNameCondition	Invalid Street Name Search Condition Was Provided
InvalidStreetTypeCondition	Invalid Street Type Search Condition Was Provided
InvalidStreetDirCondition	Invalid Street Direction Search Condition Was Provided

InvalidAltStrNameCondition	Invalid Alt. Street Name Search Condition Was Provided
InvalidAltStrTypeCondition	Invalid Alt. Street Type Search Condition Was Provided
InvalidAltStrDirCondition	Invalid Alt. Street Direction Search Condition Was Provided
InvalidMncCondition	Invalid Municipality Search Condition Was Provided
InvalidProvinceCondition	Invalid Province Search Condition Was Provided

---

## ALT\_MessageCategory Enumeration

Alternative Address Message Category enumeration represents error categories of errors detected during address analysis, related to specific address alternatives given.

Enumeration	Description
ALT_MajorError	Major Address Analysis Error
ALT_MinorError	Minor Address Analysis Error
ALT_OptimizationError	Address Optimization Error

---

## ALT\_MessageDetail Enumeration

Alternative Address Message Detail enumeration represents error details of errors detected during address analysis, related to specific address alternatives given.

Enumeration	Description
ALT_TypeMissing	Address Type Missing
ALT_TypeWrong	Address Type Wrong
ALT_TypeMisspelled	Address Type Misspelled
ALT_TypeSurplus	Address Type Surplus
ALT_WrongDelimiters	Problem With Delimiters
ALT_SuffixSpace	Problem With Suffix Space
ALT_InvalidAltValue	Invalid Alternate Value
ALT_RouteInfoSeparated	Route Information In Form RR1
ALT_DirectionMissplaced	Misplaced Direction
ALT_RouteInfoMissplaced	Misplaced Route Information
ALT_MissplacedInfo	Misplaced Info In General
ALT_OtherIrregularities	Other Irregularities

ALT_CorrectionNoSpace	Correction Failed – No Space
ALT_CorrectionNoLines	Correction Failed – No Lines
ALT_CorrectionFormatting	Correction Failed – Formatting
ALT_CorrectionInternal	Correction Failed – Internal Error
ALT_OptimizationInternal	Optimization Failed – Internal Error
ALT_IndexInternalError	Optimization Failed – Index Error
ALT_PCConflict	Postal Code Conflict
ALT_MultipleStreetsPerPC	Multiple Streets Per Postal Code
ALT_DeliveryModeChange	Delivery Mode Change Not Allowed
ALT_TooManyChangedElements	Too Many Changed Elements
ALT_TypeCannotBeChanged	Type Cannot Be Changed

---

## CStyleCategory Enumeration

Correction Style Category enumeration represents categories of address style changes that are the result of address correction and optimization. CStyleCategory enumeration is new to nCodeWS4.

Enumeration	Description
CSTYLE_CAT_ADDRESS	Address Changed Category
CSTYLE_CAT_STREET_TYPE	Street Type Category
CSTYLE_CAT_STREET_DIRECTION	Street Direction Category
CSTYLE_CAT_UNIT_KEYWORD	Unit Keyword Category
CSTYLE_CAT_STATION_TYPE	Station Type Category
CSTYLE_CAT_POBOX_KEYWORD	PO Box Keyword Category
CSTYLE_CAT_ROUTE_KEYWORD	Route Information Category
CSTYLE_CAT_GD_KEYWORD	GD Keyword Category
CSTYLE_CAT_PROVINCE	Province Keyword Category
CSTYLE_CAT_UNIT_INFO_FORMAT	Unit Information Format Category
CSTYLE_CAT_LETTERCASE_FORMAT	Letter Case Category
CSTYLE_CAT_POSTAL_CODE_FORMAT	Postal Code Category
CSTYLE_CAT_LINE_SHIFT_FORMAT	Line Shifting Category

CSTYLE_CAT_ACCENTED_LETTERS	Accented Letters Category
CSTYLE_CAT_MUNICIPALITY	Municipality Name Category
CSTYLE_CAT_STREET	Street Name Category

---

## CStyleChangeType Enumeration

Correction Style Change Type enumeration represents types of address style changes that are the result of address correction and optimization. CStyleChangeType enumeration is new to nCodeWS4.

Enumeration	Description
CSTYLE_TYPE_ERROR	Unexpected Error
CSTYLE_TYPE_NOT_APPLICABLE	Correction Style Not Applicable
CSTYLE_TYPE_CORRECTED_TO_OPT	Corrected To Optimum
CSTYLE_TYPE_CORRECTED_TO_ALT	Corrected To Alternate
CSTYLE_TYPE_ADDRESS_CHANGED	Address Was Changed
CSTYLE_TYPE_ADDRESS_NO_CHANGES	Address Was Not Changed
CSTYLE_TYPE_OPT_VALUE_FORCED	Optimum Value Was Forced
CSTYLE_TYPE_ALT_VALUE_FORCED	Alternate Value Was Forced
CSTYLE_TYPE_OPT_VALUE_KEPT	Optimum Value Was Kept
CSTYLE_TYPE_ALT_VALUE_KEPT	Alternate Value Was Kept
CSTYLE_TYPE_EMPTY_VALUE_KEPT	Empty Value Was Kept
CSTYLE_TYPE_UNIT_HYPHEN_FORCED	Unit Info Hyphen Format Forced
CSTYLE_TYPE_UNIT_KW_FORCED	Unit Info Keyword Format Forced
CSTYLE_TYPE_UNIT_HYPHEN_KEPT	Unit Info Hyphen Format Kept
CSTYLE_TYPE_UNIT_KW_KEPT	Unit Info Keyword Format Kept
CSTYLE_TYPE_UNIT_CORRECTED_TO_KW	Unit Info Corrected To Keyword Form
CSTYLE_TYPE_UNIT_CORRECTED_TO_HYPHEN	Unit Info Corrected To Hyphen Form
CSTYLE_TYPE_UPPERCASE_FORCED	Upper Case Format Was Forced
CSTYLE_TYPE_LOWERCASE_FORCED	Lower Case Format Was Forced
CSTYLE_TYPE_UPPERCASE_KEPT	Upper Case Format Was Kept
CSTYLE_TYPE_LOWERCASE_KEPT	Lower Case Format Was Kept
CSTYLE_TYPE_PC_SPACE_FORCED	Postal Code Space Format Was Forced



CSTYLE_TYPE_PC_NOSPACE_FORCED	Postal Code No Space Format Was Forced
CSTYLE_TYPE_PC_SPACE_KEPT	Postal Code Space Format Was Kept
CSTYLE_TYPE_PC_NOSPACE_KEPT	Postal Code No Space Format Was Kept
CSTYLE_TYPE_PC_CORRECTED_TO_SPACE	Postal Code Corrected To Space Format
CSTYLE_TYPE_PC_CORRECTED_TO_NOSPACE	Postal Code Corrected To No Space Format
CSTYLE_TYPE_LINE_SHIFT_NO	No Line Shifting Was Detected
CSTYLE_TYPE_LINE_SHIFT_UP	Lines Were Shifted Up
CSTYLE_TYPE_LINE_SHIFT_DOWN	Lines Were Shifted Down
CSTYLE_TYPE_ACCENTS_FORCED	Accented Format Was Forced
CSTYLE_TYPE_NO_ACCENTS_FORCED	No Accents Format Was Forced
CSTYLE_TYPE_ACCENTS_KEPT	Accented Format Was Kept
CSTYLE_TYPE_NO_ACCENTS_KEPT	No Accents Format Was Kept

---

## DAB\_ResultType Enumeration

Direct Address Browsing Result Type enumeration represents types of output produced by address browsing.

Enumeration	Description
DAB_Addresses	Address List Was Created
DAB_Names	Name List Was Created
DAB_Empty	Resulting List Is Empty

---

## KW\_CPCDBPresent Enumeration

Keyword Search CPC DB Presence enumeration indicates whether a keyword was found in the CPC database.

Enumeration	Description
KW_CPCDBPresentYes	Keyword Was Used In CPC Provided DB
KW_CPCDBPresentNo	Keyword Was Not Used In CPC Provided DB

---

## KW\_Language Enumeration

Keyword Search Language enumeration represents the language condition on keyword searches.

Enumeration	Description
KW_LanguageE	English Keywords Only

KW_LanguageF	French Keywords Only
KW_LanguageAll	Both English And French Keywords

---

## KW\_OptimumOnly Enumeration

Keyword Search Optimum Only enumeration represents the optimum indicator condition on keyword searches.

Enumeration	Description
KW_OptimumOnlyYes	Optimum Keywords Only
KW_OptimumOnlyNo	Both Optimum And Other Keywords

---

## NAT\_AddressType Enumeration

Address Types enumeration represents types of address elements that can be found in Canadian addresses.

Enumeration	Description
NAT_STRNAME	Street Name
NAT_STRTYPE	Street Type
NAT_STRDRCTN	Street Direction
NAT_STRNBR	Street Number
NAT_UNITKW	Unit Keyword
NAT_UNITNBR	Unit Number
NAT_STNNAME	Station Name
NAT_STNTYPE	Station Type
NAT_STNQLFR	Station Qualifier
NAT_POBOXKW	PO Box Keyword
NAT_POBOXNBR	PO Box Number
NAT_ROUTENBR	Route Number
NAT_ROUTEKW	Route Keyword
NAT_GENDELKW	General Delivery Keyword
NAT_MNC	Municipality
NAT_PROV	Province
NAT_PC	Postal Code
NAT_COUNTRY	Country

NAT_EXTRAINFO	Extra Information
NAT_FREEFORM	Free Address Format
NAT_ADRESSSLINE	Address Line Format
NAT_ADDRESSEE	Addressee Type
NAT_NONADDRESS	Non Address Type
NAT_STATUS	Address Status Type

Please note that the Free Address Format type includes all the address types 1 through 19. Similarly, Address Line Format includes all address types 1 through 19, excluding Municipality, Province and Postal Code.

---

## NAA\_AnalysisType Enumeration

Type of Address Analysis enumeration represents types of address analysis supported by nCode. Please note that Postal Code Lookup is obsolete and is listed here for backwards compatibility reasons.

Enumeration	Description
ADDRESS_VALIDATION	Address Validation
ADDRESS_CORRECTION	Address Correction
POSTAL_CODE_LOOKUP	Postal Code Lookup

---

## NAM\_ChangedKeyElements Enumeration

Maximum Number of Changed Key Address Elements enumeration is used to specify how many key address elements can be changed with address still remaining correctable.

Enumeration	Description
NAM_CHANGED_KEY_NONE	No Changes To Key Address Elements
NAM_CHANGED_KEY_ONE	Maximum One Key Address Element Changed
NAM_CHANGED_KEY_TWO	Maximum Two Key Address Element Changed
NAM_CHANGED_KEY_THREE	Maximum Three Key Address Element Changed

---

## NAM\_MissingRouteInfoMode Enumeration

Penalize Missing Route Information In Rural Street Served By Route Addresses Mode enumeration. Not penalizing missing route information can be useful in address capture scenarios. NAM\_MissingRouteInfoMode enumeration is new to nCodeWS4.

Enumeration	Description
NAM_MRI_PENALIZE_YES	Penalize Missing Route Info
NAM_MRI_PENALIZE_NO	Do Not Penalize Missing Route Info

---

## NB\_AllInRange Enumeration

Native Search Number Range Inclusion enumeration specifies whether database addresses found must have their number ranges fully or only partially satisfy the search range condition.

Enumeration	Description
NB_AllInRangeNo	DB Address Found Must Have At Least One Number In The Range
NB_AllInRangeYes	DB Address Found Must Have All Its Numbers In The Range

---

## NB\_IsIncluded Enumeration

Native Search Inclusion Criteria enumeration determines whether a certain search category is included in or excluded from the search.

Enumeration	Description
NB_IncludedNo	Exclude The Applicable Category
NB_IncludedYes	Include The Applicable Category

---

## NB\_TargetType Enumeration

Native Search Target Type enumeration defines the result type of the search. NB\_TargetPostalCodes was added with nCode 12.1.

Enumeration	Description
NB_TargetAddresses	Search For Addresses
NB_TargetNames	Search For Street / Station Names
NB_TargetMunicipalities	Search For Municipalities
NB_TargetFSAs	Search For FSA-s
NB_TargetPostalCodes	Search For Postal Codes

---

## NAM\_AcceptUnit enumeration

The enumeration for accepting unit info not found in PCAD.

Enumeration	Description
NAM_ACCEPT_UNIT_NO	Do not Accept Unit Info if not Found in PCAD
NAM_ACCEPT_UNIT_YES	Accept Unit Info Even if not Found in PCAD

---

## NAM\_PropertyFlag Enumeration

Property Flag On or Off enumeration.

Enumeration	Description
-------------	-------------

NAM_PROPERTY_OFF	The Specified Property Is Off
NAM_PROPERTY_ON	The Specified Property Is On

---

## NAM\_SpellingLevel Enumeration

Spelling Level of Address Analysis enumeration.

Enumeration	Description
NAM_SPELLING_LEVEL_ONE	Spelling Level One
NAM_SPELLING_LEVEL_TWO	Spelling Level Two
NAM_SPELLING_LEVEL_THREE	Spelling Level Three
NAM_SPELLING_LEVEL_FOUR	Spelling Level Four
NAM_SPELLING_LEVEL_FIVE	Spelling Level Five

---

## NAM\_MissingWordsMode Enumeration

Missing Words Mode enumeration lists options for guessing missing words in street names. NAM\_MissingWordsMode enumeration is new to nCodeWS4.

Enumeration	Description
NAM_GUESS_MW_NEVER	Never attempt to guess missing words
NAM_GUESS_MW_PCNO_MNCYES	Attempt to guess missing words only if postal code was missing and municipality was present
NAM_GUESS_MW_PCANY_MNCYES	Attempt to guess missing words only if municipality was present, regardless of postal code
NAM_GUESS_MW_ALWAYS	Always attempt to guess missing words

---

## NAM\_SOACompliant Enumeration

Analysis Mode SOA Compliancy enumeration. NAM\_SOACompliant enumeration is new to nCodeWS4.

Enumeration	Description
NAM_SOA_COMPLIANT_YES	The Analysis Mode Is SOA Compliant
NAM_SOA_COMPLIANT_NO	The Analysis Mode Is Not SOA Compliant

---

## NCS\_Lettercase Enumeration

Address Optimization Letter Case Style enumeration.

Enumeration	Description
NCS_UPPERCASE	Force Uppercase

NCS_LOWERCASE	Force Lowercase
NCS_SAMECASE	Keep The Same Letter Case

---

## NCS\_Accent Enumeration

Address Optimization Accented Letter Style enumeration.

Enumeration	Description
NCS_SAMEACCENT	Keep The Same Style
NCS_FORCEACCENT	Force Accented Output
NCS_FORCENOACCENT	Force Non-Accented Output

---

## NCS\_UnitForm Enumeration

Address Optimization Unit Information Style enumeration.

Enumeration	Description
NCS_SAMEUNIT	Keep The Same Unit Info Style
NCS_FORCEHYPHEN	Force The Hyphen Unit Info Style
NCS_FORCEKEYWORD	Force The Keyword Unit Info Style

---

## NCS\_Punctuation Enumeration

Address Optimization Punctuation Style enumeration.

Enumeration	Description
NCS_KEEPPUNCTUATION	Keep The Existing Punctuation
NCS_REMOVEPUNCTUATION	Remove The Punctuation

---

## NCS\_PCForm Enumeration

Address Optimization Postal Code Style enumeration.

Enumeration	Description
NCS_PCSAMESTYLE	Keep The Same Postal Code Style
NCS_PCFORCESPACE	Force Space In Postal Code
NCS_PCFORCENOSPACE	Force No Space In Postal Code

---

## NCS\_ExtraInfo Enumeration

Address Optimization Extra Information Style enumeration.

Enumeration	Description
NCS_EIDONOTCHANGE	Do Not Change Placement Of Extra Information
NCS_EIOPTIMIZE	Optimize Placement Of Extra Info

---

## NCS\_KWOptimum Enumeration

Address Optimization Keyword Optimum Style enumeration.

Enumeration	Description
NCS_KWDONOTCHANGE	Keep Keywords In Existing Form
NCS_KWOPTIMIZE	Convert Keywords To DB Values

---

## NCS\_MSOptimum Enumeration

Address Optimization Misspellings Optimum Style enumeration.

Enumeration	Description
NCS_MSCLOSEST	Change To Closest Valid Value
NCS_MSOPTIMIZE	Change To DB Value

---

## NCS\_PushLines Enumeration

Address Optimization Push Lines Style enumeration. NCS\_PushLines enumeration is new to nCodeWS4.

Enumeration	Description
NCS_PUSH_LINES_UP	Push Lines Up
NCS_PUSH_LINES_DOWN	Push Lines Down

---

## NCS\_StreetNameOptimum Enumeration

Address Optimization Street Name Optimum Style enumeration. NCS\_StreetNameOptimum enumeration is new to nCodeWS4.

Enumeration	Description
NCS_KEEP_EXISTING_STREET	Keep The Existing Street Name, Even If Alternate Street Name Was Used
NCS_FORCE_OFFICIAL_STREET	Force The Official Street Name Provided By CPC

---

## NCS\_MunicipalityNameOptimum Enumeration

Address Optimization Municipality Name Optimum Style enumeration. NCS\_MunicipalityNameOptimum enumeration is new to nCodeWS4.

Enumeration	Description
-------------	-------------

NCS_KEEP_EXISTING_MNC	Keep The Existing Municipality Name, Even If Alternate Municipality Name Was Used
NCS_FORCE_OFFICIAL_MNC	Force The Official Municipality Name Provided By CPC

---

## NCS\_SOACompliant Enumeration

Correction Style SOA Compliancy enumeration. NCS\_SOACompliant enumeration is new to nCodeWS4.

Enumeration	Description
NCS_SOA_COMPLIANT_YES	Correction Style Is SOA Compliant
NCS_SOA_COMPLIANT_NO	Correction Style Is Not SOA Compliant

---

## NPAR\_ParameterType Enumeration

Parameter Type enumeration.

Enumeration	Description
NPAR_AddressLayouts	List Of Address Layouts
NPAR_AnalysisModes	List Of Analysis Modes
NPAR_CorrectionStyles	List Of Correction Styles

---

## NAM\_LanguageMode Enumeration

The language mode of address analysis enumeration.

Enumeration	Description
NAM_LANGUAGE_MODE_ENGLISH	The language mode of address analysis is english.
NAM_LANGUAGE_MODE_FRENCH	The language mode of address analysis is french.







# Appendix A – Address Status

---

## Address Status

When performing address analysis, be it address correction or address validation, nCode will determine the address status of the analyzed address:

- primary address status reflects the official SERP rules for address correction and address validation
- secondary address status reflects nCode's internal categorization of addresses
- urban address status determines whether the address is considered urban or rural

---

## Primary Address Status

The primary address status can have the following values:

Status	Description
V	The address is declared valid.
I	The address is declared invalid.
C	The address is declared correctable.
N	The address is declared non-correctable.
F	The address is declared foreign.

Note that in the case of address validation, only valid and invalid status can be reported. In the case of address correction, only valid, correctable, non-correctable and foreign status values apply. The address validation status of invalid is equivalent to the address correction status of correctable, non-correctable or foreign.

---

## Secondary Address Status

The secondary address status can have the following values:

Status	Description
	Space is the default secondary address status. It has no special meaning.
A	Apartment Address - Questionable as per SERP Handbook.

B	Apartment Address. No unit range in reference data. Unit provided in input.
C	Apartment Address. Unit range in reference data.
D	Deliver to Post Office address. PO Box number range is between 99900 and 99905.
K	Valid LVR Address (Street, PO Box or GD) with no significant errors.
L	Incomplete LVR Address with proper LVR postal code in input.
M	LVR corrected by application of proper postal code (and other elements, optionally).
O	Complete address with accurate rural postal code.
P	Incomplete address, corrected to rural postal code.
Q	Correctable rural address with non-postal code elements corrected by nCode.
R	Rural Address - Questionable as per SERP Handbook.
S	Complete Civic Rural Address.
T	Correctable Civic Rural Address.
Y	Questionable unit number in the input address was not found in the address database. PCAD only.

Note that the secondary statuses 'D', 'S' and 'T' are new to nCode 11 and nCodeWS4. The secondary status 'Y' is new to nCode 12.2.

---

## Urban Address Status

The urban address status can have the following values:

Status	Description
U	The address is declared urban.
R	The address is declared rural.

# Appendix B – Address Lines

---

## Address Lines

Addresses are passed to and received from nCodeWS4 Web Service as arrays of address lines. Every address line consists of address line name and address line content. Address lines much match the [address layout](#) being used. When passing address lines in nCodeWS4 Web Service requests, please note the following:

- The names of the address lines used to define the requests are case sensitive and must match the names of address lines in the selected address layout.
- If some of the address lines that exist in the address layout were not listed in the request, they will be presumed to be empty.
- If an address line were passed in the request and its name did not match the address layout in question, it would be ignored by nCodeWS4.
- The order in which the address line name/address line content pairs are provided is not important.
- If the address line content exceeds the length of the address line, it will be truncated.

When receiving address lines in nCodeWS4 Web Service responses, all address lines defined in the address layout will be present, even if their content was empty.

---

## Address Line Example

For example, address layout named 'ADDR x2 LAST LINE' has the address record length of 165 and three address lines. Each address line is defined with its address line name, relative address line start, address line length and the address types that belong to the address line, as follows:

Name	Start	Length	Address Line Types
Address1	1	50	Address Line
Address2	51	50	Address Line
LastLine	101	60	Municipality, Province, Postal Code

Here is an example of a valid nCodeWS4 Web Service request in the given address layout:

Name	Content
Address1	125 Commerce Valley Dr W Suite 200

LastLine      Thornhill ON L3T 7W4

# Glossary of Terms

## **Address Browsing**

A feature of nCode that allows you to search for lists of valid addresses or valid address elements that match search criteria which you provide. Compare *Address Searching*.

## **Address Capture**

A feature of nCode that allows you to complete an address with as little typing as possible. As you type, nCode provides a list of closest alternatives you can select from. nCode's guessing algorithm can anticipate the intended address based on just a few elements entered, like street number and street name, postal code, etc.

## **Address Correction**

A feature of nCode that allows you to take an address of unknown validity and to analyze this address, looking for valid information, and resulting in either a fully validated address or a list of valid addresses which are likely valid alternatives to the given invalid address. nCode can be configured to automatically correct addresses or you can choose to have nCode provide messages indicating which corrections are required.

## **Address Element**

A standard component of an address as recognized by Canada Post. Examples include street number, street name, street type, municipality, province, and so on. Most address elements are very specific, but some can be more general. For example, the Free Form address element is a placeholder for all known address elements. An address line with the Free Form element scoped to it can have any kind of address information on it. Similarly, the Address Line address element is a placeholder for all address elements other than Municipality, Province, and Postal Code.

## **Address Layout**

Parameters in nCode that define which address elements appear as part of an address and on which address line and in which position they will occur. You can define any number of arbitrary address layouts as your business needs require. Address layouts consist of a single string of a known length which is subdivided into any number of Address Lines.

## **Address Line**

A part of an address layout which consists of a string of known length which begins at a known position within the address layout. Each address line has one or more address elements scoped to it. Scoping address elements to address lines is a way of giving hints to the nCode analysis engine regarding where to look for different types of address elements. It also tells nCode where you expect to see the various address elements in an optimized address.

## **Address Optimization**

A feature of nCode which allows you to take a valid or correctable address and to format it according to parameters which you define.

## **Address Searching**

A feature of nCode that allows you to find address information at various levels, including FSA, municipality, street, and postal code range using a variety of specific search criteria. Compare *Address Browsing*.

## **Address Transformation**

A feature of nCode which allows you to take a valid or correctable address formatted according to one Address Layout and to reformat this address according to a different Address Layout.

## **Alternative Address**

A valid address or postal code address range that could potentially match the input address information provided to nCode. When there is one valid alternative address and there are no missing or incorrect delivery details which preclude the identification of a specific delivery point, nCode will deem an address to be valid. If there are multiple equally valid alternative addresses for a given input then nCode will deem the address to be non-correctable.

## **Analysis Mode**

A group of settings that define how an address will be analyzed by nCode. For example, an Analysis Mode specifies whether accented letters are to be expected in the address records, whether address elements ought to be separated by blanks, which address elements must not be changed when performing address correction and many others.

## **API**

Application Program Interface. A set of procedures within a program that can be used by other programs to perform specific tasks. The nCode API contains procedures which let your programs perform address analysis, correction, searching, and transformation.

## **Azimuth**

An azimuth, meaning "a way, a part, or quarter," is an angular measurement in a spherical coordinate system. The n from an observer (origin) to a point of interest is projected perpendicularly onto a reference plane; the angle between the projected vector and a reference vector on the reference plane is called the azimuth. Azimuth is usually measured in degrees (°). The concept is used in navigation, astronomy, engineering, mapping, mining and artillery.

## **Block Face Data**

See *PCAD*.

## **Browsing**

see *Address Browsing*.

## **Correctable Address**

An address which is not valid but which has enough correct information to allow nCode to recognize the valid address which is implied by the invalid address.

## **Correction**

see *Address Correction*.

## **Correction Style**

A group of settings that define how an address will be corrected or optimized. For example, a Correction Style will determine whether or not the corrected/optimized address record will have: letters forced into uppercase or lowercase, accented letters (when applicable), a space in the postal code and so on.

## **CORS**

Cross-origin resource sharing (CORS) is a mechanism that allows many resources (e.g., fonts, JavaScript, etc.) on a web page to be requested from another domain outside the domain the resource originated from. In particular, JavaScript's AJAX calls can use the XMLHttpRequest mechanism. Such "cross-domain" requests would otherwise be forbidden by web browsers, per the same origin security policy. CORS defines a way in which the browser and the server can interact to determine whether or not to allow the cross-origin request. It is more useful than only allowing same-origin requests, but it is more secure than simply allowing all such cross-origin requests.

## **Declared Valid**

According to Canada Post SERP rules, rural and LVR addresses are to be declared valid if they have a valid rural or LVR postal code. In a declared valid address, the non-postal code information is not to be changed according to SERP



rules, even if this information is not the official delivery information according to Canada Post's postal code data reference file.

In order to maintain its SERP certification, nCode must conform to the rules for declaring rural and LVR addresses valid. In some cases, users of nCode may wish to circumvent these rules. Rural and LVR addresses that are merely declared valid (but which are not complete and accurate) can be detected by an analysis status of "V" with a valid alternative count of "0". Users of nCode may use this condition to detect a declared valid address. Using nCode's address searching features with the (validated) rural or LVR postal code, the official information for that postal code can be retrieved.

### **Foreign Address**

An address which is clearly not a valid Canadian postal address but which appears to be a foreign (typically U.S.) address will be deemed to be foreign by nCode.

### **FSA**

Forward Sortation Area. A geographical subset of Canada used by Canada Post. The first three characters in a postal code indicate the FSA. Note that FSA<sup>OM</sup> is an official mark of Canada Post Corporation.

### **Invalid Address**

An address which is not valid and which does not have enough correct information to allow nCode to suggest a reasonable number of potential valid alternative addresses that correspond to the input information.

### **Invalid Excluded**

Starting with the release of Point of Call (Range Based) Address Data in January 2011, Canada Post has introduced a new classification of addresses. This classification only applies during a transitional period (provisionally January to July 2011). Invalid excluded addresses are non-correctable but are not counted against the accuracy score of a mailing on the Statement of Accuracy report. For an address to be categorized as Invalid Excluded, it must be for a postal code that is being managed by Canada Post at the point of call level and it must have invalid range information. For practical purposes, this means that the address is for an apartment building but the provided apartment number is not in one of the known valid ranges for that building.

### **JSON**

JSON, or JavaScript Object Notation, is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML. Although originally derived from the JavaScript scripting language, JSON is a language-independent data format, and code for parsing and generating JSON data is readily available in a large variety of programming languages.

### **Key Address Element**

An address element which has been flagged as being more reliable than others for analysis purposes. Under Canada Post's SERP rules, all address elements are to be considered equally weighted when analyzing an address. In some cases, such as online, real-time address data capture, it is desirable to ignore this SERP rule and to give greater credence to some address elements than others. By designating an address element as key in the analysis mode, users of nCode can indicate that they have above average confidence in the data for that address element. When analyzing an address, nCode uses the maximum number of key address elements setting to prevent consideration of possible alternatives that would require changes to too many key address elements.

### **Layout**

see *Address Layout*.

### **LDU**

Local Delivery Unit. A subset of a postal code used by Canada Post to indicate a specific set of delivery points within a postal code. The last three characters in a postal code indicate the LDU. Note that LDU<sup>OM</sup> is an official mark of Canada Post Corporation.

## **LVR**

Large Volume Receivers are those Canada Post customers who are receiving more than one hundred pieces of mail per day and who generally have a dedicated, unique postal code.

## **Non-Correctable Address**

An address which is not valid and which does not have enough correct information to allow nCode to recognize the valid address which is implied in the input information. Non-Correctable addresses may have multiple valid alternatives which nCode will list. nCode will deem an address non-correctable if there are two or more equally likely valid alternatives for the given input information. nCode will also deem an address non-correctable if the supplied delivery point information is inconsistent with the only likely valid alternative. For example, if a street number of 10 is given in the input, but the address that matches the remaining input information requires a street number in the range of 100-200, then nCode will deem the address non-correctable.

## **PCAD**

Postal Code Address Data Product. This is a data file published by Canada Post that contains all of the valid mailing addresses in Canada. The data in the PCAD file is at the range level, sometimes called the block face level. This means that each postal code is represented as one or more ranges, for example: even numbers from 10 to 78 NEW ST in HAMILTON, ON, L8P 4J4. This data is available to be licensed from Canada Post for both commercial and non-commercial purposes. It is updated each month. nCode uses PCAD data for interactive operations through the nCode API. nCode stores PCAD data in the NCODEREF.DAT file. Compare *PoCAD*.

## **PoCAD**

Point of Call (Range Based) Address Data Product. This is a data file published by Canada Post that contains all of the valid mailing addresses in Canada. The data in the PoCAD file is range based, like the PCAD file, but it is more granular, with many postal codes being managed at the point of call level, meaning that one can be sure that if a delivery point is within the range, it is a real delivery point and not, for example, an empty lot. PoCAD data has only been made available by Canada Post since January 2011. It is available only for use by commercial mailing applications which are recognized by Canada Post's SERP program. According to Canada Post's usage restrictions, this data can only be used in batch mode by a SERP-recognized program for the purpose of creating a Statement of Accuracy Report for a mailing that is intended to be delivered by Canada Post. This data is updated each month. nCode uses PoCAD data for SOA generation through the nCode Batch Processor and the batch processing feature of the nCode Configuration Utility. nCode stores PoCAD data in the NCODEPOC.DAT file. Compare *PCAD*.

## **Point of Call Data**

See *PoCAD*.

## **Postal Code**

A six-character alphanumeric combination comprised of an FSA and an LDU that is assigned to one or more postal addresses in a specific delivery area. It is an integral part of every postal address in Canada. Note that Postal Code<sup>OM</sup> is an official mark of Canada Post Corporation.

## **REST**

REST stands for Representational State Transfer. REST is an architecture style for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines. RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.

## **SERP**

Software Evaluation and Recognition Program. A testing program conducted by Canada Post which evaluates and rates the accuracy of address correction software. nCode is fully certified by Canada Post under the SERP Address Accuracy category.

## SOA

See *Statement of Accuracy*. SOA is also a common abbreviation for Service-Oriented Architecture, which, according to Wikipedia, "is a flexible set of design principles used during the phases of systems development and integration in computing. A system based on a SOA will package functionality as a suite of interoperable services that can be used within multiple separate systems from several business domains." nCode Web Services Edition is ideally suited to integration into applications built using a service-oriented architecture.

### Statement of Accuracy

The Statement of Accuracy Report (often: SOA) is a report produced by nCode and other SERP recognized software which must be provided to Canada Post along with a mailing to certify the accuracy of a mailing's addresses. A Statement of Accuracy Report is one of the requirements for obtaining postage discounts from Canada Post. The report contains statistics regarding a mailing list, in particular the number of records and the overall percentage of those records which are valid for mailing. Other statistics include numbers of questionable rural and apartment addresses. The specifications for the Statement of Accuracy Report are set out in the SERP Handbook, as published by Canada Post.

### Transformation

see *Address Transformation*.

### Valid Address

An address which is substantially complete and correct and requires no changes, other than perhaps some cosmetic ones in order to be complete and correct from Canada Post's perspective. See also: *Declared Valid*.

### Vincenty

Two formulas, or more precisely, iterative numerical methods, for solving the direct and indirect geodetic problems, which calculate the distance between two points on the surface of a spheroid. These methods are much more accurate when working with points on the Earth's surface than using so-called Great Circle calculations. Named for its inventor, Thaddeus Vincenty.

## WCF

The Windows Communication Foundation (or WCF) is a runtime and a set of APIs (application programming interface) in the .NET Framework for building connected, service-oriented applications. WCF is a tool often used to implement and deploy a service-oriented architecture (SOA). It is designed using service-oriented architecture principles to support distributed computing where services have remote consumers. Clients can consume multiple services; services can be consumed by multiple clients. Services are loosely coupled to each other. Services typically have a WSDL interface (Web Services Description Language) that any WCF client can use to consume the service, regardless of which platform the service is hosted on. WCF implements many advanced Web services (WS) standards such as WS-Addressing, WS-ReliableMessaging and WS-Security. With the release of .NET Framework 4.0, WCF also provides RSS Syndication Services, WS-Discovery, routing and better support for REST services.



# Index

## A

address capture 2, 14, 16, 18, 87, 92  
address correction 71  
address layout 2, 7, 11, 12, 15, 16, 18, 26, 28, 70, 71, 72, 73, 74, 75, 89, 104, 111, 112, 113  
address lines 27, 109, 111  
address status 107, 108  
address types 98  
AJAX 19, 25, 112  
alternate keywords 8, 15, 16, 18  
analysis mode 7, 9, 15, 16, 18, 29, 31, 34, 70, 71, 73, 74, 75, 89, 101, 104, 112  
[analyze address](#) 14, 16, 18, 21, 22, 23, 24, 25, 30, 47, 48, 49, 60, 70  
[analyze get alternatives](#) 14, 16, 18, 21, 22, 23, 24, 25, 47, 49, 52, 71, 72

## B

[browse address](#) 15, 16, 18, 21, 22, 23, 24, 25, 34, 47, 52, 58, 74, 75

## C

C#.NET 19, 20, 22, 26, 29, 31, 33, 43  
correction style 7, 9, 11, 15, 16, 18, 29, 31, 34, 70, 71, 73, 74, 75, 89, 95, 96, 104, 112  
CORS 6, 13, 112

## D

direction 11, 53, 54, 56, 57, 64, 65, 68, 69, 76, 77, 81, 84, 85

## F

[format address](#) 15, 21, 22, 23, 24, 47, 49, 51, 52, 58, 72, 73  
[format parser elements](#) 15, 16, 21, 23, 24, 47, 52, 58, 73

## G

[get address layout](#) 15, 16, 18, 21, 22, 23, 25, 26, 27, 29, 32, 34, 47, 62, 82  
[get analysis mode](#) 15, 16, 18, 21, 22, 23, 25, 48, 63, 79  
[get correction style](#) 15, 16, 18, 21, 22, 23, 25, 26, 48, 63, 80  
[get keywords](#) 15, 16, 18, 21, 22, 23, 25, 47, 56, 81, 82  
[get parameter list](#) 15, 16, 18, 21, 22, 23, 25, 27, 47, 60, 83  
guessing modes 14, 57, 58, 59, 63, 68, 79, 86

## J

Java 1, 2, 3, 5, 6, 19, 23, 24  
jQuery 5, 19, 25  
JSON 3, 5, 6, 13, 15, 113

## N

nCode Configuration Utility 6, 9, 11, 12  
nCode Programmers Toolkit 1, 2, 4, 6, 7, 8, 9, 11, 12, 14, 112, 114  
nCode reference database 71, 72  
nCode Server 2, 5, 19, 20, 22, 23, 24  
nCode Web Service 2, 5, 6, 7, 13, 14, 15, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 31, 33, 43, 47, 55, 67, 76, 91, 108, 109  
nCode Web Service Interface 14, 15, 26  
nCode Web Services 2, 3, 4, 5, 6, 7, 9, 11, 12, 13, 19, 20, 26, 115  
nCodeWS3 2, 13, 14, 49, 76  
new to nCodeWS4 34, 44, 47, 49, 50, 53, 54, 55, 57, 59, 67, 69, 70, 72, 74, 78, 79, 80, 84, 85, 86, 87, 92, 93, 95, 96, 99, 101, 103, 104

## O

[oneStop address capture](#) 14, 16, 18, 21, 22, 23, 24, 25, 47, 59, 60, 86, 87

Web Service proxy 19, 26

WSDL 3, 5, 6, 13, 19, 115

## P

[parse address](#) 15, 16, 18, 21, 22, 23, 24, 25, 47, 57, 74

postal code 2, 8, 9, 11, 41, 43, 44, 48, 49, 50, 53, 57, 59, 64, 65, 101, 108, 111, 112, 113, 114

## R

[remove parameter](#) 15, 21, 23, 24, 47, 62, 83

REST 3, 5, 6, 13, 20, 114, 115

RESTful 2, 5, 6, 13, 15, 114

## S

[search for addresses](#) 15, 16, 18, 21, 22, 23, 24, 25, 45, 47, 52, 53, 54, 57, 76, 78, 100

[search for alt. municipalities](#) 15, 21, 22, 23, 24, 47, 54, 84

[search for alt. streets](#) 15, 21, 22, 23, 25, 47, 55, 85

[search for names](#) 15, 16, 18, 21, 22, 23, 24, 25, 47, 54, 57, 76, 78

SERP 1, 10, 11, 107, 108, 112, 113, 114, 115

[set address layout](#) 15, 21, 23, 24, 48, 64, 80, 83

[set analysis mode](#) 15, 21, 23, 24, 48, 64, 79, 80

[set correction style](#) 15, 21, 23, 24, 48, 65, 80

SOAP 2, 3, 5, 6, 13, 14, 15, 20, 114

source code examples 26

source code samples 19

## T

[transform address](#) 14, 16, 18, 21, 22, 23, 24, 25, 32, 47, 51, 60, 75

## V

VB.NET 3, 6

## W

WCF 2, 5, 13, 20, 26, 115