Alere™ i Instrument
Data Connectivity Specification
JSON File Format

This document is intended for Trial-Use only, and has not been officially released for use with the Alere™ i Instrument.

Title	Alere™ i Instrument, Data Connectivity Specification, JSON File Format
Document Number	856-999-14-R&D-S0041
Revision	v0.1
Date	4-Nov-2014

1 Introduction

This document provides a specification of the test results data file format used by the Alere™ i instrument running current software.

Instrument test results are saved in “Java Serialized Object Notation” format referred to as a *JSON file*.

Alere™ i instruments save test results as JSON files in the internal non-volatile memory. These files can be accessible to users via several methods. Typically these include:

- **Export:** Test results can be exported to a USB Flash Memory Key attached to the Alere™ i Instrument.
- **Network Data Connection:** Where the Alere™ i instrument is set up with external network access where a test result list and individual files are available at a specific URL location built from the IP address of the instrument.

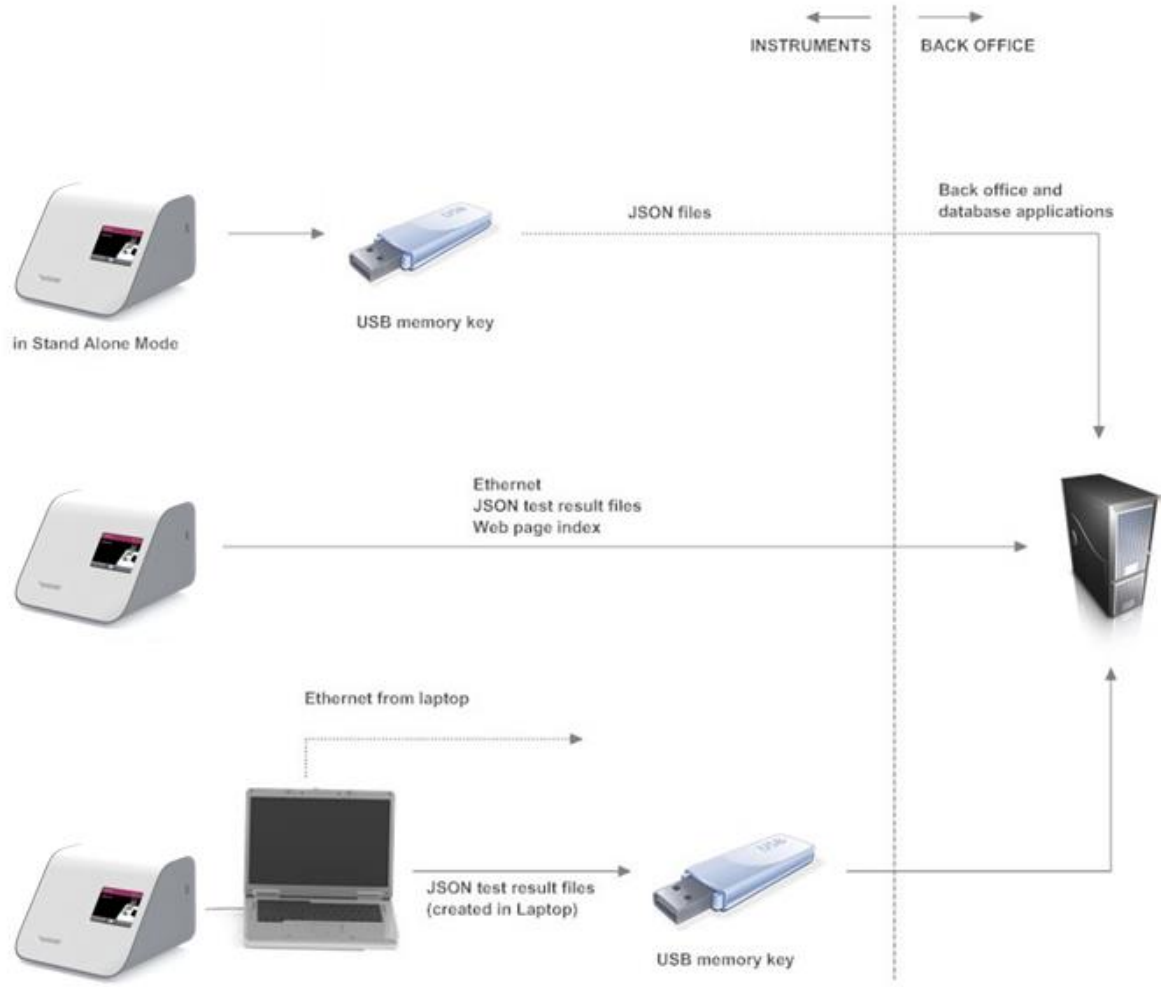


Figure 1: JSON File Modes

2 Scope

This document describes the schema used by the latest release of software only.

The schema defined in this revision of the document is current for the instrument software. The schema and format of the files may change in future.

3 JSON File Description

3.1 File Format

Test results are saved as serialised objects in JSON format.

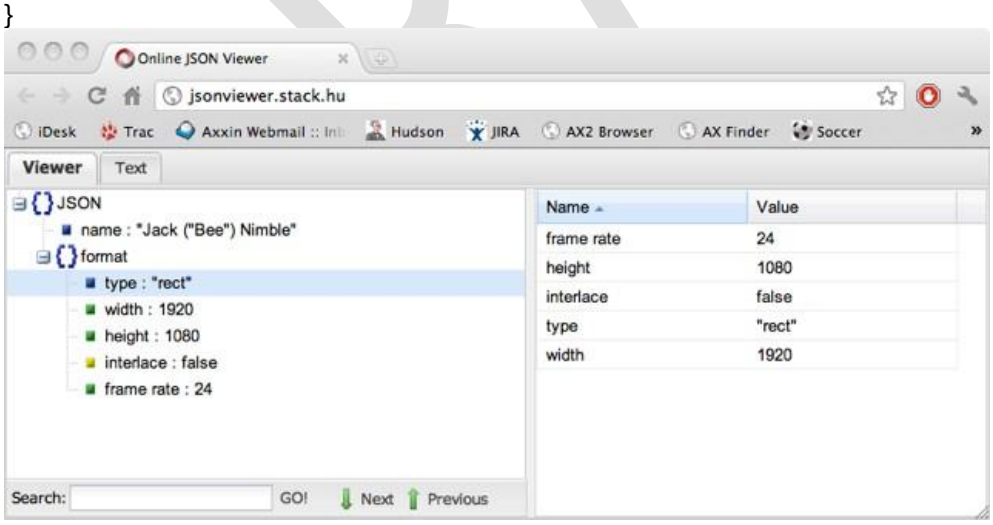
"JSON format, or JavaScript Object Notation, is a text-based open standard designed for human-readable data interchange. It is derived from the JavaScript scripting language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for many languages. Java format referred to as JSON (JavaScript Object Notation).This is a lightweight data-interchange format based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language."[1]

For further details on JSON system description refer to: *http://json.org/*

3.2 JSON File Viewers

A number of generic JSON file viewer application are available. The following example shows the text of a JSON file and the representation in a third online JSON Viewer.

```
{
   "name": "Jack (\"Bee\") Nimble",
   "format": {
      "type":      "rect",
      "width":     1920,
      "height":    1080,
      "interlace":  false,
      "frame rate": 24
   }
}
```



JSON Text viewed in a Free online JSON Viewer: http://jsonviewer.stack.hu/

1 www.wikipedia.org

3.3 File Types

Test results are stored on the instrument in a ‘compact’ format. These files are generated at the point where the test result is saved on the instrument.

The compact test results are exported from the instrument by either the web server or to a USB key using the ‘Export Results’ function. The full test results are exported only using the ‘Export Logs’ feature of the instrument – these files are intended for service use.

Two types of test result file exist, a Patient Test Result and a QC Test Result. Both file types have the same schema, but are named differently.

3.4 Privacy and Patient ID

The compact test results are intended for the end user and contain the Patient ID – it is not blanked prior to creation.

QC test results do not have this field blanked.

3.5 File Naming

Files are named using the instrument serial number, time, date and a GUID (Global Unique Identifier) that is also contained as fields inside the serialized file.

Both the full and compact results have the same naming format. A

patient test result has the following format:

```
Alerei_Patient_<timestamp>_<serial>_<id>.json
```

A QC test result has the following format:

```
Alerei_QC__<timestamp>_<serial>_<id>.json
```

Where:

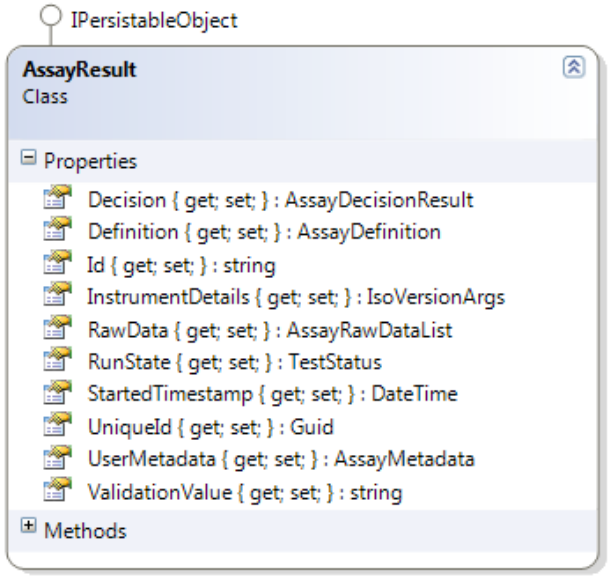
- `<timestamp>` is in the format `YY-MM-DD_HH-MM-SS` where `HH` is in 24 hour time.
- `<serial>` is the instrument serial number.
- `<id>` is the test result id (a GUID).

3.6 Hash Function

Each test result contains a calculated MD5 hash string. This number allows an external system to confirm that the file has been transferred correctly without transmission error or tampering. Information on how the check value is calculated is detailed in section 7.

4 Test Result File Schema Definition

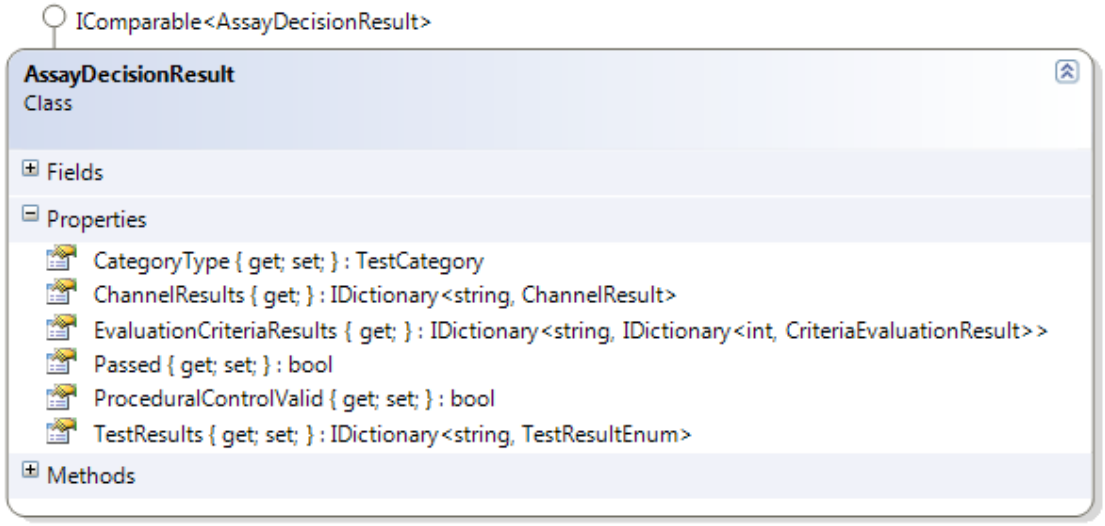
The file is a serialized Assay Result object from the source code. The design of this class defines the file format.

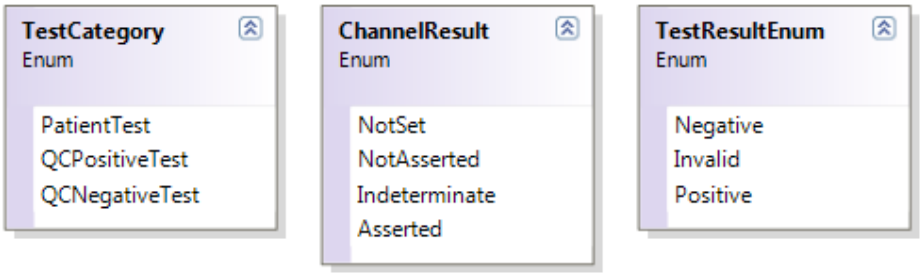


FIELD NAME	FIELD TYPE	DESCRIPTION
Decision	AssayDecisionResult	The decision outcome for the test, see the definition of the AssayDecisionResult type below.
Definition	AssayDefinition	The definition of the assay, see the definition of the AssayDefinition type below.
Id	string	A field used by the 'store' in the instrument software to index the result only. It is not the GUID for the test result or the 'test number'. This field is not for external use.
InstrumentDetails	IsoVersionArgs	Information about the instrument that performed the test. Refer to the definition of the IsoVersionArgs type below.
RawData	AssayRawDataList	The Raw data that made up the test, much of this data is nulled in the compact result. Refer to the definition of the AssayRawDataList below.
RunState	TestStatus	An internal field used to keep track of the state of the result as the result is created. Not for external use. TestStatus is a simple enumeration, see below for the definition of this enumeration.
StartedTimestamp	DateTime	The time the test was started. This is a C# DateTime type.
UniqueId	Guid	A unique GUID identifier that is generated for every test. The GUID is represented as a string in the JSON.
UserMetadata	AssayMetadata	Matadata about the test. See the definition of the AssayMetadata type below.
ValidationValue	string	The check value (as generated by the instrument when the file is created). This is used to check against file corruption. Details of the method used to calculate this value can be found in section 7

4.1 Assay Decision Result Type Schema

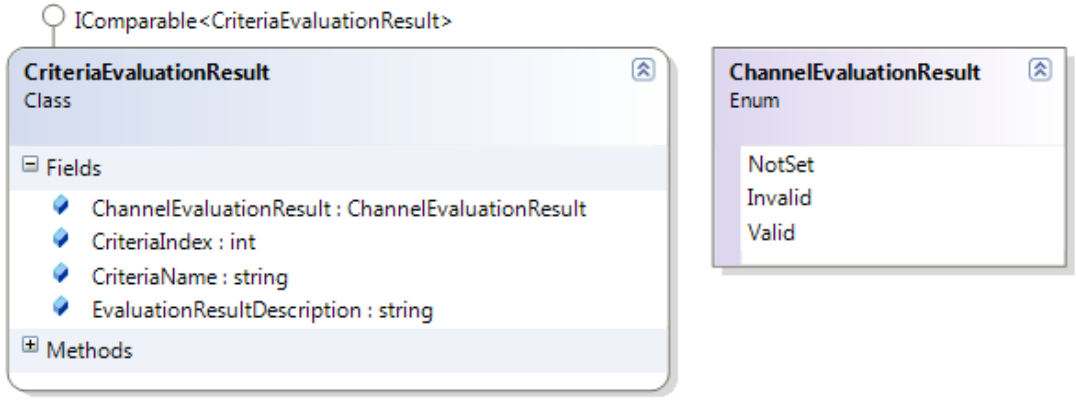
This type is contained within the overall Assay Result and contains information about the results of the decision algorithm used to relate raw results to the outcome of the test.





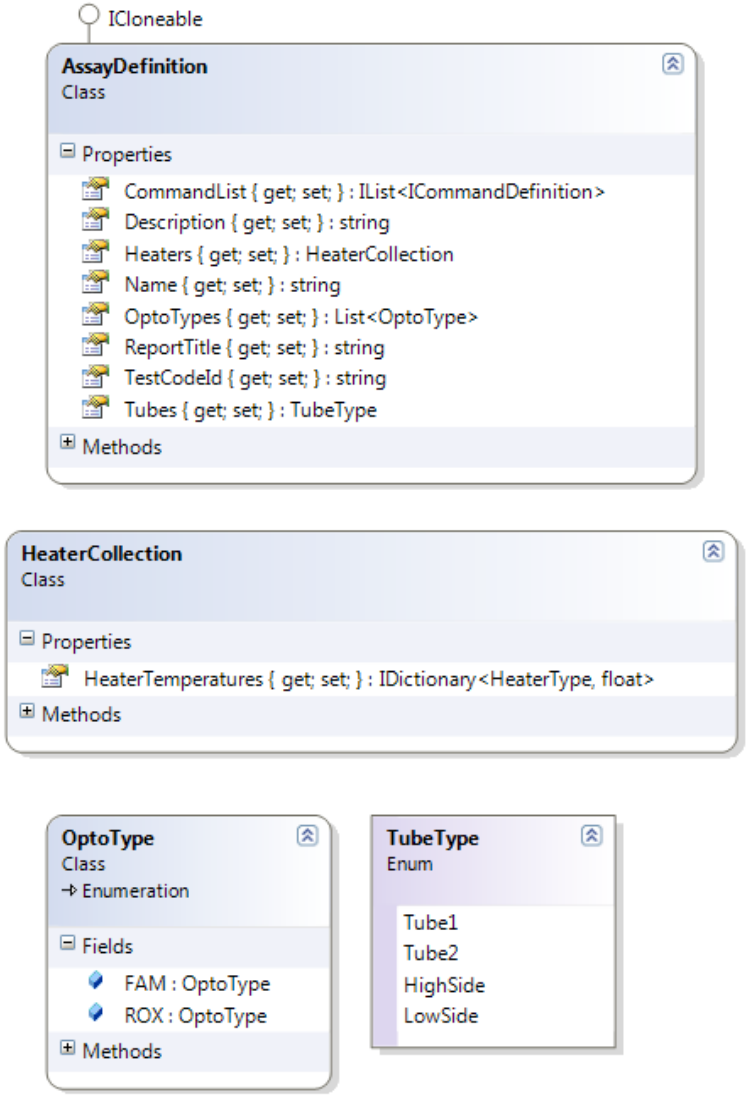
FIELD NAME	FIELD TYPE	DESCRIPTION
CategoryType	TestCategory	The decision outcome for the test, see the definition of the AssayDecisionResult type below.
ChannelResults	ChannelResult	The definition of the assay, see the definition of the AssayDefinition type below.
EvaluationCriteriaResults	IDictionary<string,IDictionary<int, CriteriaEvaluationResult>>	A two dimensional key value pair list of evaluation criteria results.
Passed	bool	Information about the instrument that performed the test. Refer to the definition of the IsoVersionArgs type below.
ProceduralControlValid	Bool	A Boolean flag indicating if the algorithm found the procedural control to be valid for the test.
TestResults	IDictionary<string, TestResultEnum>	A key value pair list of the interpreted test results by the algorithm.

4.1.1 Criteria Evaluation Result Type Schema



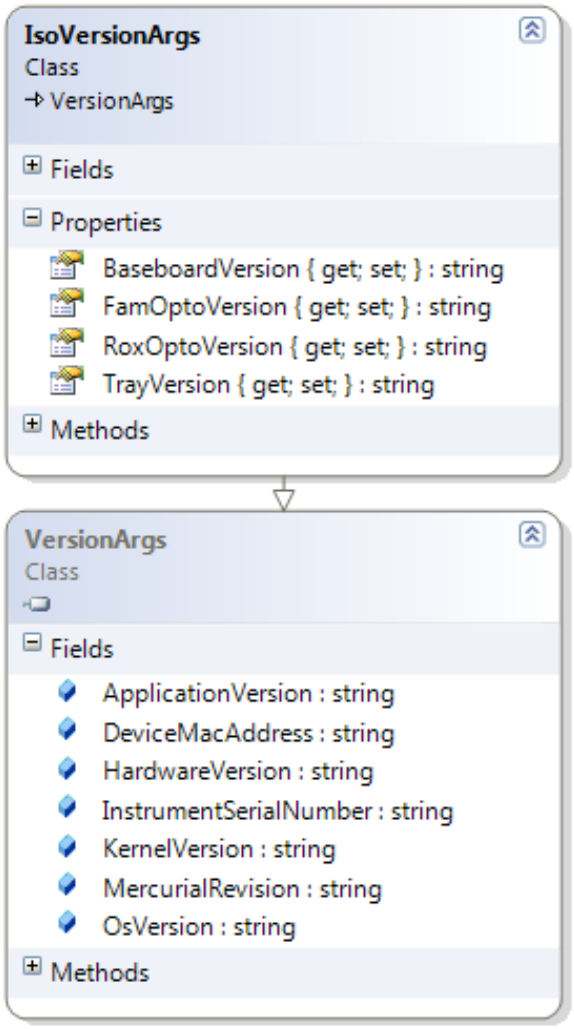
FIELD NAME	FIELD TYPE	DESCRIPTION
`ChannelEvaluationResult`	`ChannelEvaluationResult`	The result for the channel, NotSet, Valid or Invalid
`CriteriaIndex`	`int`	An index number for the criteria result assigned by the instrument software
`CriteriaName`	`string`	A name of the criteria as a string
`EvaluationResultDescription`	`string`	A description of the evaluation result.

4.2 Assay Definition Type Schema



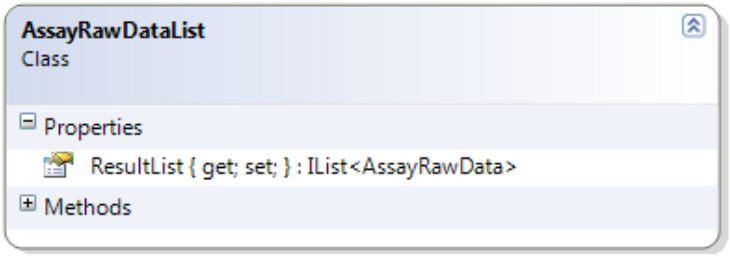
FIELD NAME	FIELD TYPE	DESCRIPTION
CommandList	IList<ICommandDefinition>	A list of the actions performed during the test, mix, read, move cycles and the like. For more information refer to the SDS document. For internal use only, typically nulled in a compact test result.
Description	string	A description of the assay type
Heaters	HeaterCollection	A list of heater set point temperatures for the test. For internal use only.
Name	string	A name of the assay type.
OptoTypes	List<OptoTypes>	A list of optical modules that are read in each read cycle block.
ReportTitle	String	The title text string to use in the printed report for the test type.
TestCodeId	String	An identifier used for a particular test type. This is the test type as used in the consumable QR code and is 02 for the Influenza test on Alere™ i
Tubes	TubeType	A list of tube positions to read in each recipe read block.

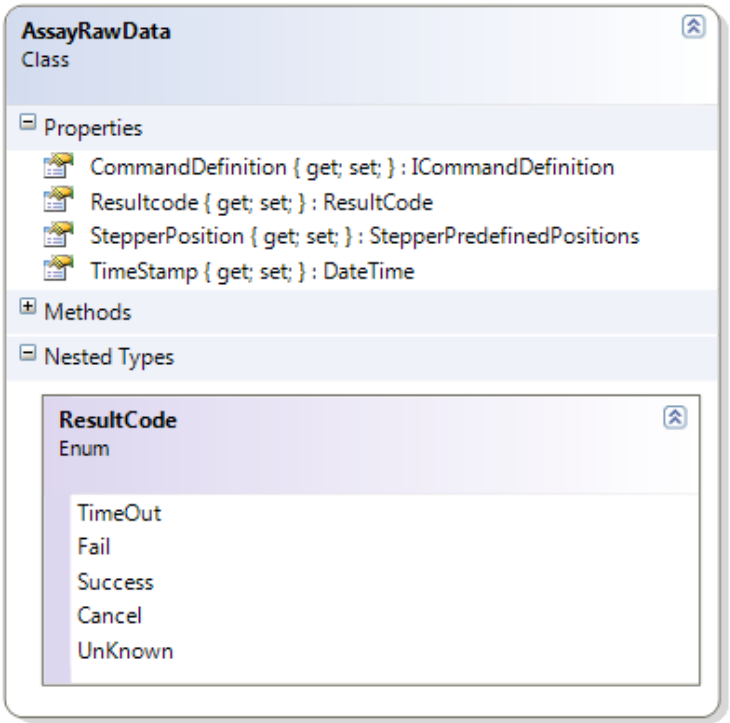
4.3 IsoVersionArgs Type Schema



FIELD NAME	FIELD TYPE	DESCRIPTION
ApplicationVersion	string	Version number of the instrument application
DeviceMacAddress	string	The mac address of the Ethernet NIC on the instrument
HardwareVersion	string	The instrument hardware version
InstrumentSerialNumber	string	The instrument serial number
KernelVersion	string	The version of the linux kernel on the instrument
MercurialVersion	String	A unique identifier for the source that generated a record that can be tied back to the instrument software version control system,
OsVersion	String	Deprecated. Not used in Stage 1.
BaseboardVersion	string	Version of the baseboard firmware at the time of the test
FamOptoVersion	string	Version of the FAM optical module firmware at the time of the test
RoxOptoVersion	string	Version of the ROX optical module firmware at the time of the test
TrayVersion	string	Version of the trayboard firmware at the time of the test

4.4 AssayRawDataList Type Schema

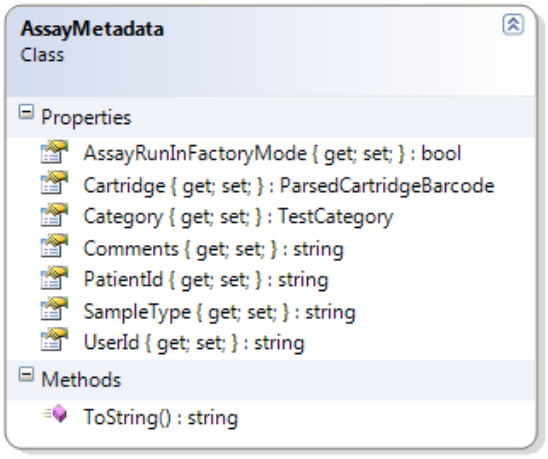


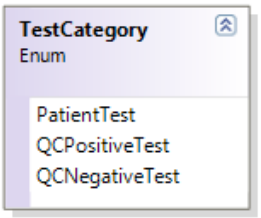


FIELD NAME	FIELD TYPE	DESCRIPTION
CommandDefinition	ICommandDefinition	Reserved. The command definition type that generated the result.
Resultcode	ResultCode	A result code for the operation. Indicates if the result is valid.
StepperPosition	StepperPredefinedPositions	The position of the rotor when the result was generated
TimeStamp	DateTime	The timestamp of the reading.

4.5 AssayMetadata Type Schema

Provides additional information about the test itself that is not related to the readings or the decision. Information about the logged in user, the sample/patient ID, the scanned QR code etc is contained in this type.





FIELD NAME	FIELD TYPE	DESCRIPTION
AssayRunInFactoryMode	bool	A flag to indicate if the test was run in factory mode
Cartridge	ParsedCartridgeBarcode	The QR code value for the consumable
Category	TestCategory	The test category, Patient or QC (with type)
Comments	string	May contain comments. Not used at present.
PatientId	String	The entered Patient/Sample ID entered by the instrument operator. *Note: This information is not included in the 'full' test result JSON file. It is blanked prior to creation of the file. The full test result is intended for supply to the manufacturer for instrument diagnostics. For privacy reasons, no personally identifiable information may be contained in these files.*

UserId	string	The ID of the logged in user that performed the test.
SampleType	string	A string field showing the sample type (either 'Swab' or 'VTM') used for this test.

5 Sample Result JSON File

The following is a sample 'compact' JSON file. It is provided for reference and to illustrate the schema shown in section 4. Formatting and line numbers has been applied for clarity. Not all data is present in this example, some values have been blanked to make overall structure readable, however the results is a valid instrument QC test result file.


```
1. {
2.   "UserMetadata": {
3.     "AssayRunInFactoryMode": false,
4.     "Category": 1,
5.     "PatientId": "2_QC",
6.     "Cartridge": null,
7.     "UserId": "user",
8.   "Comments": null
9.   },
10. "RawData": {
11.     "ResultList": {
12. "$type":
    "System.Collections.Generic.List`1[[Iso.Domain.Result.AssayRawData,
    Iso.Domain]], mscorlib",
13.       "$values": [
14.         {
15.           "TimeStamp": "\/Date(1373003756877+1000)\/",
16.           "Resultcode": 2,
17.           "StepperPosition": 14,
18.           "CommandDefinition": null
19.         }
20.       ]
21.     }
22.   },
23. "Decision": {
24.     "TestResults": {
25. "$type":       "System.Collections.Generic.Dictionary`2[[System.String,
    mscorlib],[Iso.Domain.AssayDecision.Enums.TestResultEnum,  Iso.Domain]],
    mscorlib",
26.       "Flu A": 1,
27.       "Flu B": 1
28.     },
29. "ChannelResults": {
30. "$type": "System.Collections.Generic.Dictionary`2[[System.String,
    mscorlib],[Iso.Domain.AssayDecision.Enums.ChannelResult, Iso.Domain]],
    mscorlib"
31.     },
32. "EvaluationCriteriaResults": {
33. "$type": "System.Collections.Generic.Dictionary`2[[System.String,
    mscorlib],[System.Collections.Generic.IDictionary`2[[System.Int32,
    mscorlib],[Iso.Domain.AssayDecision.CriteriaEvaluationResult,    Iso.Domain]],
    mscorlib]], mscorlib"
34.     },
35.     "CategoryType": 1,
36.     "Passed": true,
37.   "ProceduralControlValid": false
38.   },
39. "Definition": {
40.     "Name": null,
41.     "Description": null,
42.     "ReportTitle": null,
43.     "TestCodeId": null,
44.     "HeaterTemperatures": {
45. "$type":
    "System.Collections.Generic.Dictionary`2[[Iso.Domain.Enums.HeaterType,
    Iso.Domain],[System.Single, mscorlib]], mscorlib"
46.     },
47. "CommandList": {
48. "$type":
```

```
    "System.Collections.Generic.List`1[[Iso.Domain.Commands.ICommandDefinition,
    Iso.Domain]], mscorlib",
49.       "$values":
[ 50.
51.       ]
52.     },
53.     "Tubes": 0,
54.     "OptoTypes":
[ 55.
56.     ]
57.   },
58.   "UniqueId": "893b1885-4762-4c99-824a-1b8546ee434f",
59.   "StartedTimestamp": "\/Date(1373003756877+1000)\/",
60.   "InstrumentDetails": {
61.     "InstrumentSerialNumber": "",
62.     "HardwareVersion": "",
63.     "ApplicationVersion": "",
64.     "DeviceMacAddress": "",
65.     "KernelVersion": "",
66.     "UpdateNumber": "",
67.     "MercurialRevision": "",
68.     "OsVersion": "",
69.     "FamOptoVersion": "",
70.     "RoxOptoVersion": "",
71.     "TrayVersion": "",
72.   "BaseboardVersion": ""
73.   },
74.   "ValidationValue": "D6A96FEC14C077039960F804829C7944",
75.   "RunState": 0,
76.   "Id": null
77.   }
```


6 Network Data Connection

Alere™ i instruments save JSON format files as the native file format for a test report. The content of the file may be adopted to be project-specific but the file structure is described in Section 2 of this document.

The JSON test result files saved within the instrument may be accessed over an Ethernet network connection if this function forms part of the project specification.

This section describes access to the file list and individual test result files on an Alere™ i Instrument.

6.1 File List

The test results are accessed as a file list of each test result in a machine readable web page.

6.2 IP Address

The instrument IP address is required to access the instrument test results web page address. For this reason, instruments connected to middleware should use a fixed IP address. Typically this IP address would relate to the instruments physical location and can be applied by the administrator user.

6.3 Network Access

A separate test result file is provided for each test. Test results are “read only” and can be read over a network connection, but cannot be deleted or altered.

The capability to access test results over a HTTP network connection is provided. This allows other software systems such as middleware applications to remotely interrogate the instrument and access test results for transfer to other systems, automatic archive or analysis.

A web browser can be used to manually access the test file set on the instrument to confirm this

function. For a specific instrument the files are accessible at the following network location:

http://172.16.0.125/cgi-bin/testresults.cgi, where “172.16.0.125” is the IP address of the specific instrument in this example. Typically the instrument will be configured with a fixed IP address when this function is used.

The IP address of the instrument can be discovered on the Network Settings Screen.

7 MD5 Hash File Check Value

Within the `AssayResult` domain object a field exits within the `AssayResult` object that stores an MD5 hash value of certain fields within the test result object. This serves as a corruption and tamper protection mechanism. When test results are saved by the store this check value is determined and stored in the ‘`ValidationValue`’ field.

This value is determined whenever one of the stores saves either a patient or QC test result. To calculate the value a string is built from the following fields concatenated:

- The GUID for the test result
- The `TestStarted` filed converted to GMT in the format of “yyyyMMddHHmmss”
- The `RunState` field which is an enumeration of the following options: `NotRun, Running, CompletedSuccessfully, Failed.`
- The ‘`AssayRunInFactoryMode`’ Field as ‘true’ or ‘false’.
- The `UserId` field
- The `PatientId` field
- The `Definition.Name` and
- The `Definition.TestCodeId` field
- A list of the `Decision.TestResult` Key Names concatenated with the Value as a string.
- The `ProceduralControlValid` field as ‘true’ or ‘false’.

This string is then encoded as an array of ASCII Bytes. An MD5 hash algorithm is then applied to the string of bytes. This value is the `ValidationValue` field.

In C# the following code snippet will calculate the validation value

```
StringBuilder sb = new StringBuilder();
sb.Append(UniqueId);
sb.Append(StartedTimestamp.ToUniversalTime().ToString("yyyyMMddHHmmss"));
sb.Append(RunState);
if (UserMetadata != null)
{
    sb.Append(UserMetadata.AssayRunInFactoryMode);
    sb.Append(UserMetadata.UserId);
    sb.Append(UserMetadata.PatientId);
}
if (Definition != null)
{
    sb.Append(Definition.Name);
    sb.Append(Definition.TestCodeId);
}
if (Decision != null)
{
    if (Decision.TestResults != null)
        foreach (KeyValuePair<string, TestResultEnum> pair in
        Decision.TestResults)
        {
            sb.Append(pair.Key);
            sb.Append(pair.Value);
        }
    sb.Append(Decision.ProceduralControlValid);
}
MD5 md5 = MD5.Create();
byte[] hash = md5.ComputeHash(Encoding.ASCII.GetBytes(sb.ToString()));
StringBuilder returnValue = new StringBuilder();
foreach (byte b in hash)
    returnValue.Append(b.ToString("X2"));
return returnValue.ToString();
```

Figure 2. Data Check Value Calculation Code Snippet (as at v2.1.4.0)

8 Description of GUID format

The term GUID is used for Microsoft's implementation of the Universally Unique Identifier (UUID) standard.

The value of a GUID is represented as a 32-character hexadecimal string, such as {21EC2020-3AEA-1069-A2DD-08002B30309D}, and is usually stored as a 128-bit integer. The total number of unique keys is 2^{128} or 3.4×10^{38}. This number is so large that the probability of the same number being generated randomly twice is negligible.

Basic Structure:

The GUID is a 16-byte (128-bit) number. The most commonly used structure of the data type is:

Bits	Bytes	Description	Endianness
32	4	Data1	Native
16	2	Data2	Native
16	2	Data3	Native
64	8	Data4	Big

Data4 stores the bytes in the same order as displayed in the GUID text encoding (see below), but the other three fields are reversed on little-endian systems (for example Intel CPUs).

One to three of the most significant bits of the second byte in Data 4 define the type variant of the GUID:

Pattern	Description
0xx	Network Computing System backward compatibility
10x	Standard
110	Microsoft Component Object Model backward compatibility; this includes the GUIDs for important interfaces like IUnknown and IDispatch
111	Reserved for future use

The most significant four bits of Data3 define the version number, and the algorithm used.

Text Encoding:

A GUID is most commonly written in text as a sequence of hexadecimal digits grouped into five groups, such as:

3F2504E0-4F89-11D3-9A0C-0305E82C3301
This text notation contains the following fields, separated by hyphens:

Hex digits	Description
8	Data1
4	Data2
4	Data3
4	Initial two bytes from Data4
12	Remaining six bytes from Data4

For the first three fields, the most significant digit is on the left. The last two fields are treated as eight separate bytes, each having their most significant digit on the left, and they follow each other from left to right. Note that the digit order of the fourth field may be unexpected, since it's treated differently than in the structure.

Often braces are added to enclose the above format, as such:

{3F2504E0-4F89-11D3-9A0C-0305E82C3301}

When printing fewer characters is desired, GUIDs are sometimes encoded into

a base64 or ASCII85 string. A base64-encoded GUID consists of 22 to 24 characters (depending on padding), for instance:

7QDBkvCA1+B9K/U0vrQx1A
7QDBkvCA1+B9K/U0vrQx1A==

and ASCII85 encoding gives 20 characters, for example:

5:$Hj:Pf\4RLB9%kU\Lj

In Uniform Resource Names (URN), the v1 GUIDs have namespace identifier "uuid", e.g.:

urn:uuid:3F2504E0-4F89-11D3-9A0C-0305E82C3301

Use of the GUID ensures that each test result is unequally identified. The file format also included the instrument ID and time and date of the test within the file name string.