Document filename: CDAInstanceChecker_UserGuide.docx			
Directorate / Programme	*DS&P*	**Project**	*Interoperability Specifications*
Document Reference		**NPFIT-FNT-TO-DSD-229**	
Project Manager	*Kevin Sprague*	**Status**	*Draft*
Owner	*Richard Kavanagh*	**Version**	*0.5*
Author	*Prashant Trivedi*	**Version issue date**	*24th Feb 2014*

CDA Instance Checker (User Guide)

Revision History

Version	Date	Summary of Changes
0.1	*17/02/2012*	*First draft for comment*
0.2	*28/02/2012*	*Updated to include:* *Checking of wrapped CDA instances* *Rendering of CDA instances* *Revised validation report format*
0.3	*16/04/2012*	*Change for Template Lookup to check templateId*
0.4	*09/04/2013*	*Updated to new document template*
0.5	*24/02/2014*	*Updated section 4.2 for input directories*

Reviewers

This document must be reviewed by the following people:

Reviewer name	Title / Responsibility	Date	Version

Approved by

This document must be approved by the following people: author to indicate approvers

Name	Signature	Title	Date	Version

Glossary of Terms

Term / Abbreviation	What it stands for

Document Control:

The controlled copy of this document is maintained in the HSCIC corporate network. Any copies of this document held outside of that area, in whatever format (e.g. paper, email attachment), are considered to have passed out of control and should be checked for currency and validity.

Contents

1 About this Document

1.1 Purpose of Document

The purpose of this document is to provide guidance on the installation and use of the CDA Instance Checker utility.

1.2 Audience

This document has been written for any individual or organisation that has a requirement to check the validity of one or more CDA instances that have been produced in accordance with an HSCIC Domain Message Specification (DMS).

1.3 Content

This document comprises this following sections / topics:

- Introduction to the CDA Instance Checker
- Installation Instructions
- Usage

2 Introduction

2.1 Overview

The CDA Instance Checker is a utility for checking the validity of one or more CDA instances represented in XML.

The core CDA Instance Checker functionality takes a directory of on-the-wire CDA instances as input and:

- Validates each on-the-wire instance against a given XML Schema and Schematron document
- Transforms each on-the-wire instance to templated format
- Validates each templated instance against a given XML Schema and Schematron document
- Produces an HTML validation report detailing the validity of the on-the-wire and templated instances (i.e. Steps 1 and 3 above respectively)
- In addition to the core functionality, the CDA Instance Checker may also be used to:
- Extract the CDA payload from an HL7 wrapped message prior to commencing Step 1 above
- Render each CDA instance as HTML to enable manual validation of the instance

2.2 Assumptions

It is assumed that:

- The CDA Instance Checker is to be used in a Windows operating system environment. The utility as described in this document assumes a given directory structure and naming convention. Given that the utility relies heavily on Java technologies it should be possible to use the utility in other operating systems (e.g. Mac OS, Linux), however instructions are not provided within this document on how to achieve this.

2.3 System Requirements

System requirements for use of CDA Instance Checker utility:

- Operating System - Windows XP or later (please refer to Assumptions section above for other operating systems)
- Java Runtime Environment – JRE 1.5.0 or later

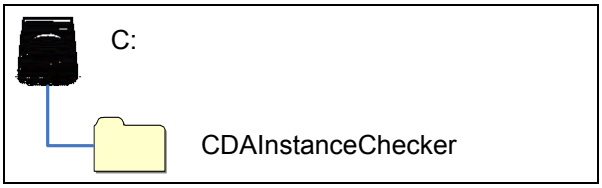
3 Installation Instructions

3.1 Check Java Runtime Environment version

Check the Java Runtime Environment (JRE) version on your machine. The CDA Instance Checker is compatible with JRE 1.5.0 or later.

3.2 Download CDA Instance Checker

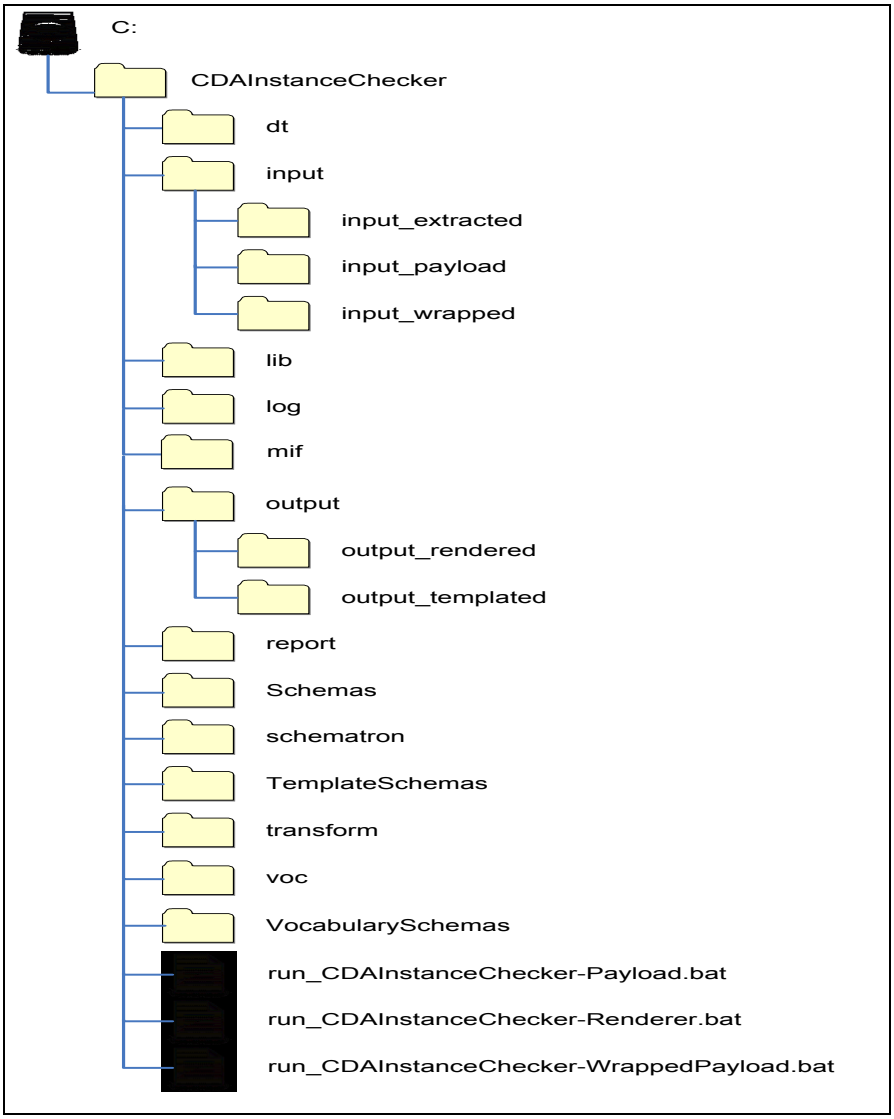
Download the CDA Instance Checker utility from tooling pack to your machine. The CDA Instance Checker utility may be installed in any location on your machine but for the purposes of this document is assumed to be installed in the root of the C*:* drive as follows:



4 Usage

4.1 CDA Instance Checker directory structure

Following installation of the CDA Instance Checker the following directory structure should be present on your machine:



The purpose of each directory is explained in the following table[1]:

Directory	Description
dt	*Contains data type XML schemas from DMS*
input\input_extracted	*Contains the on-the-wire CDA payloads created by the CDA Instance Checker following stripping the HL7 wrappers from the instances in the input\input_wrapped directory*
input\input_payload	*Contains the on-the-wire CDA instances that act as input to the CDA Instance Checker*
input\input_wrapped	*Contains the wrapped on-the-wire CDA instances that act as input to the CDA Instance Checker. These instances will be stripped of their HL7 wrappers and the CDA payloads written to the input\input_extracted directory*
lib	*Contains library resources used by the CDA Instance Checker*
log	*Contains the log file maintained by the CDA Instance Checker*
mif	*Contains the .mif files from DMS*
output\output_rendered	*Contains the CDA instances rendered as HTML by the CDA Instance Checker*
output\output_templated	*Contains the templated CDA instances created by the CDA Instance Checker*
report	*Contains the validation reports generated by the CDA Instance Checker*
Schemas	*Contains XML Schemas used to validate CDA instances from DMS*
schematron	*Contains XML schematron validating stylesheets used by the CDA Instance Checker*
TemplateSchemas	*Contains XML Schemas used to validate CDA instances from DMS*
transform	*Contains XSLT stylesheets used by the CDA Instance Checker*
voc	*Contains vocabulary XML Schemas from DMS*
VocabularySchemas	*Contains vocabulary XML Schemas from DMS*

4.2 Copying the DMS content

Copy the following directories from the DMS into the *CDAInstanceChecker* directory:

Schemas

TemplateSchemas

Mif

voc

[1] Directories with italicised names are empty upon installation of the CDA Instance Checker. They should be populated by copying content directly from the directory of the same name within the Domain Message Specification whose instances are being validated

dt

VocabularySchemas

4.3 Populating the input directories

Populate the *input\input_payload* directory with the on-the-wire CDA instances OR the *input\input_wrapped* directory with the wrapped CDA instances to be validated by the CDA Instance Checker.

Populate the *mif* directory with mif files from domain message specification along with other directories as listed in section 4.2.

4.4 CDA Instance Checker batch files

The CDA Instance Checker includes 3 batch files:

- *run_CDAInstanceChecker-Payload.bat* – used to validate a directory of on-the-wire CDA instances
- *run_CDAInstanceChecker-WrappedPayload.bat* – used to validate a directory of wrapped CDA instances
- *run_CDAInstanceChecker-Renderer.bat* – used to render a directory of CDA instances as HTML

4.4.1 The *run_CDAInstanceChecker-Payload* batch file

The *run_CDAInstanceChecker-Payload* batch file contains commands to run the following tasks: 0

- Create TemplateIdLookup.xml in schematron folder file to check templateId usage in on-the-wire CDA instances.
- Validate the on-the-wire CDA instances against an XML Schema and Schematron
- Transform the on-the-wire CDA instances to templated CDA instances
- Validate the newly created templated CDA instances against an XML Schema and Schematron
- Transform the validation reports to HTML
- Open the HTML validation reports

The *run_CDAInstanceChecker-Payload* batch file utilises the following variables whilst running these tasks:

Variable	Description
%input_mif%	*Directory where all mif files are copied from domain message specification*
%xsl_templatelookup%	*XSLT file used to create TemplateIdLookup.xml*
%output_templatelookup%	*Directory where TemplateIdLookup.xml file is created.*

%input_payload%	*Directory of on-the-wire CDA instances to be used as input for the CDA Instance Checker*
%output_templated%	*Directory where templated CDA instances will be written following transformation*
%schematron_otw%	*Schematron validating stylesheet to be applied to each on-the-wire instance*
%schematron_templated%	*Schematron validating stylesheet to be applied to each templated instance*
%xsl_transform%	*XSLT file to use to transform an on-the-wire instance to a templated instance*
%xsl_render%	*XSLT file to use to render the validation report as HTML*
%report_xml_otw%	*On-the-wire instances XML validation report to be produced by the CDA Instance Checker*
%report_xml_templated%	*Templated instances XML validation report to be produced by the CDA Instance Checker*
%report_html_otw%	*On-the-wire instances HTML validation report to be produced by the CDA Instance Checker*
%report_html_templated%	*Templated instances HTML validation report to be produced by the CDA Instance Checker*
%log%	*Log file maintained by the CDA Instance Checker*
%schema_otw%	*XML schema to be explicitly applied to each on-the-wire instance. If this variable has a value of "null" then the CDA Instance Checker will assume that a schemaLocation attribute is present within the root element of each CDA instance*
%schema_templated%	*XML schema to be explicitly applied to each templated instance. If this variable has a value of "null" then the CDA Instance Checker will assume that a schemaLocation attribute is present within the root element of each CDA instance*

The following subsections describe the commands used to run each task within the *run_CDAInstanceChecker-Payload* batch file. For ease of reading commands to redirect the standard output and error streams to *%log%* have been omitted from these subsections. A full listing of the *run_CDAInstanceChecker-Payload* batch file is available in Section 5.1.

4.4.1.1 Step 0 – Transform to create TemplateIdLookup.xml in Schematron folder

Variables Used:	*% input_mif%, % xsl_templatelookup%, %output_templatelookup%*

Utilises Saxon 9.3 library classes to transform the directory of mif files at *%input_mif%* into TemplateIdLookup.xml using *%xsl_ templatelookup %*. The Look Up file for all templateIds used in the domain will be output to *%output_templatelookup%*.

```
java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%input_mif% -xsl:%xsl_templatelookup% -
o:%output_templatelookup%
```

4.4.1.2 Step 1 – Schema and Schematron validate the on-the-wire CDA instances

Variables Used:	*%input_payload%, %report_xml_otw%, %schematron_otw%, %schema_otw%*

Utilises the bespoke *lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar* to validate the directory of on-the-wire CDA instances[2] at *%input_payload%* against *%schema_otw%* and *%schematron_otw%*. An XML validation report will be output to *%report_xml_otw%*.

If *%schema_otw%* has the value "null" each CDA instance is assumed to specify the XML Schema that should be used to validate it within the *schemaLocation* attribute of its root element.

Xerces-J 2.9.0 is added to the classpath to ensure consistent validation results regardless of the JRE version installed on the user's machine.

```
IF %schema_otw%==null (
      java -classpath lib\xerces-
      2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-
      1_0\cdaInstanceChecker.jar
      org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker
      %input_payload% %report_xml_otw% %schematron_otw%
) ELSE (
      java -classpath lib\xerces-
      2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-
      1_0\cdaInstanceChecker.jar
      org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker
      %input_payload% %report_xml_otw% %schematron_otw% %schema_otw%
)
```

4.4.1.3 Step 2 – Transform the on-the-wire CDA instances to templated CDA instances

Variables Used:	*%input_payload%, %xsl_transform%, %output_templated%*

Utilises Saxon 9.3 library classes to transform the directory of on-the-wire CDA instances at *%input_payload%* into templated CDA instances using *%xsl_transform%*. The templated CDA instances will be output to *%output_templated%*.

```
java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%input_payload% -xsl:%xsl_transform% -
o:%output_templated%
```

4.4.1.4 Step 3 - Schema and Schematron validate the templated CDA instances

Variables Used:	*%output_templated%, %report_xml_templated%, %schematron_templated%, %schema_templated%*

[2] Any validation errors that exist within the on-the-wire CDA instances will not be corrected by the CDA Instance Checker and may have an impact on the validity of the templated CDA instances generated in Step 2

Utilises the bespoke *lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar* to validate the directory of templated CDA instances[3] at *%output_templated%* against *%schema_templated%* and *%schematron_templated%*. An XML validation report will be output to *%report_xml_templated%*.

If *%schema_templated%* has the value "null" each CDA instance is assumed to specify the XML Schema that should be used to validate it within the *schemaLocation* attribute of its root element.

Xerces-J 2.9.0 is added to the classpath to ensure consistent validation results regardless of the JRE version installed on the user's machine.

```
IF %schema_templated%==null (

      java -classpath lib\xerces-
      2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-
      1_0\cdaInstanceChecker.jar
      org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker
      %output_templated% %report_xml_templated%
      %schematron_templated%

) ELSE (

      java -classpath lib\xerces-
      2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-
      1_0\cdaInstanceChecker.jar
      org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker
      %output_templated% %report_xml_templated%
      %schematron_templated% %schema_templated%

)
```

4.4.1.5 Step 4 – Transform the XML validation reports to HTML

Variables Used:	*%report_xml_otw%, %report_xml_templated%, %xsl_render%, %report_html_otw%, %report_html_templated%*

Utilises Saxon 9.3 library classes to transform the XML validation reports *%report_xml_otw%* and *%report_xml_templated%* to the HTML validation reports *%report_html_otw%* and *%report_html_templated%* respectively using *%xsl_render%*.

```
java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%report_xml_otw% -xsl:%xsl_render% -
o:%report_html_otw%

java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%report_xml_templated% -xsl:%xsl_render% -
o:%report_html_templated%
```

[3] Any validation errors that exist within the on-the-wire CDA instances will not be corrected by the CDA Instance Checker and may have an impact on the validity of the templated CDA instances generated in Step 2

4.4.1.6 Step 5 – Open the HTML validation reports

Variables Used:	*%report_html_otw%, %report_html_templated%*

Opens the HTML validation reports *%report_html_otw%* and *%report_html_templated%*.

```
start %report_html_generic%
start %report_html_templated%
```

4.4.2 The *run_CDAInstanceChecker-WrappedPayload* batch file

The *run_CDAInstanceChecker-WrappedPayload* batch file contains commands to run the following tasks:

- Create TemplateIdLookup.xml in schematron folder file to check templateId usage in on-the-wire CDA instances.
- Extract the on-the-wire payload instances from the HL7 wrapped instances
- Validate the extracted on-the-wire CDA payloads against an XML Schema and Schematron
- Transform the on-the-wire CDA instances to templated CDA instances
- Validate the newly created templated CDA instances against an XML Schema and Schematron
- Transform the validation reports to HTML
- Open the HTML validation reports

The *run_CDAInstanceChecker-WrappedPayload* batch file utilises the following variables whilst running these tasks:

Variable	Description
%input_mif%	*Directory where all mif files are copied from domain message specification*
%xsl_templatelookup%	*XSLT file used to create TemplateIdLookup.xml*
%output_templatelookup%	*Directory where TemplateIdLookup.xml file is created.*
%input_wrapped%	*Directory of wrapped on-the-wire CDA instances to be used as input for the CDA Instance Checker*
%input_extracted%	*Directory where extracted on-the-wire payloads will be written following stripping of HL7 wrappers*
%output_templated%	*Directory where templated CDA instances will be written following transformation*
%schematron_otw%	*Schematron validating stylesheet to be applied to each on-the-wire instance*
%schematron_templated%	*Schematron validating stylesheet to be applied to each templated instance*

%xsl_extract%	*XSLT file used to extract payload from HL7 wrapped instance*
%xsl_transform%	*XSLT file to use to transform an on-the-wire instance to a templated instance*
%xsl_render%	*XSLT file to use to render the validation report as HTML*
%report_xml_otw%	*On-the-wire instances XML validation report to be produced by the CDA Instance Checker*
%report_xml_templated%	*Templated instances XML validation report to be produced by the CDA Instance Checker*
%report_html_otw%	*On-the-wire instances HTML validation report to be produced by the CDA Instance Checker*
%report_html_templated%	*Templated instances HTML validation report to be produced by the CDA Instance Checker*
%log%	*Log file maintained by the CDA Instance Checker*
%schema_otw%	*XML schema to be explicitly applied to each on-the-wire instance. If this variable has a value of "null" then the CDA Instance Checker will assume that a schemaLocation attribute is present within the root element of each CDA instance*
%schema_templated%	*XML schema to be explicitly applied to each templated instance. If this variable has a value of "null" then the CDA Instance Checker will assume that a schemaLocation attribute is present within the root element of each CDA instance*

The following subsections describe the commands used to run each task within the *run_CDAInstanceChecker-WrappedPayload* batch file. For ease of reading commands to redirect the standard output and error streams to *%log%* have been omitted from these subsections. A full listing of the *run_CDAInstanceChecker-WrappedPayload* batch file is available in Section 5.2.

4.4.2.1 Step 0 – Transform to create TemplateIdLookup.xml in Schematron folder

Variables Used:	*% input_mif%, % xsl_templatelookup%, %output_templatelookup%*

Utilises Saxon 9.3 library classes to transform the directory of mif files at *%input_mif%* into TemplateIdLookup.xml using *%xsl_ templatelookup %*. The Look Up file for all templateIds used in the domain will be output to *%output_templatelookup%*.

```
java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%input_mif% -xsl:%xsl_templatelookup% -
o:%output_templatelookup%
```

4.4.2.2 Step 1 – Extract the CDA on-the-wire payloads

Variables Used:	*%input_wrapped%, %xsl_extract%, %input_extracted%*

Utilises Saxon 9.3 library classes to transform the directory of wrapped CDA instances at *%input_wrapped%* into CDA instance payloads using *%xsl_extract%*. The CDA payload instances will be output to *%input_extracted%*.

```
java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%input_wrapped% -xsl:%xsl_extract% -
o:%input_extracted%
```

4.4.2.3 Step 2 – Schema and Schematron validate the on-the-wire CDA payloads

Variables Used:	*%input_extracted%, %report_xml_otw%, %schematron_otw%, %schema_otw%*

Utilises the bespoke *lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar* to validate the directory of on-the-wire CDA payloads[4] at *%input_extracted%* against *%schema_otw%* and *%schematron_otw%*. An XML validation report will be output to *%report_xml_otw%*.

If *%schema_otw%* has the value "null" each CDA instance is assumed to specify the XML Schema that should be used to validate it within the *schemaLocation* attribute of its root element.

Xerces-J 2.9.0 is added to the classpath to ensure consistent validation results regardless of the JRE version installed on the user's machine.

```
IF %schema_otw%==null (

      java -classpath lib\xerces-
      2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-
      1_0\cdaInstanceChecker.jar
      org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker
      %input_extracted% %report_xml_otw% %schematron_otw%

) ELSE (

      java -classpath lib\xerces-
      2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-
      1_0\cdaInstanceChecker.jar
      org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker
      %input_extracted% %report_xml_otw% %schematron_otw%
      %schema_otw%

)
```

4.4.2.4 Step 3 – Transform the on-the-wire CDA instances to templated CDA instances

Variables Used:	*%input_extracted%, %xsl_transform%, %output_templated%*

Utilises Saxon 9.3 library classes to transform the directory of on-the-wire CDA payload instances at *%input_extracted%* into templated CDA instances using *%xsl_transform%*. The templated CDA instances will be output to *%output_templated%*.

[4] Any validation errors that exist within the on-the-wire CDA instances will not be corrected by the CDA Instance Checker and may have an impact on the validity of the templated CDA instances generated in Step 3

```
java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%input_extracted% -xsl:%xsl_transform% -
o:%output_templated%
```

4.4.2.5 Step 4 - Schema and Schematron validate the templated CDA instances

Variables Used:	*%output_templated%, %report_xml_templated%, %schematron_templated%, %schema_templated%*

Utilises the bespoke *lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar* to validate the directory of templated CDA instances[5] at *%output_templated%* against *%schema_templated%* and *%schematron_templated%*. An XML validation report will be output to *%report_xml_templated%*.

If *%schema_templated%* has the value "null" each CDA instance is assumed to specify the XML Schema that should be used to validate it within the *schemaLocation* attribute of its root element.

Xerces-J 2.9.0 is added to the classpath to ensure consistent validation results regardless of the JRE version installed on the user's machine.

```
IF %schema_templated%==null (

      java -classpath lib\xerces-
      2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-
      1_0\cdaInstanceChecker.jar
      org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker
      %output_templated% %report_xml_templated%
      %schematron_templated%

) ELSE (

      java -classpath lib\xerces-
      2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-
      1_0\cdaInstanceChecker.jar
      org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker
      %output_templated% %report_xml_templated%
      %schematron_templated% %schema_templated%

)
```

4.4.2.6 Step 5 – Transform the XML validation reports to HTML

Variables Used:	*%report_xml_otw%, %report_xml_templated%, %xsl_render%, %report_html_otw%, %report_html_templated%*

Utilises Saxon 9.3 library classes to transform the XML validation reports *%report_xml_otw%* and *%report_xml_templated%* to the HTML validation reports *%report_html_otw%* and *%report_html_templated%* respectively using *%xsl_render%*.

[5] Any validation errors that exist within the on-the-wire CDA instances will not be corrected by the CDA Instance Checker and may have an impact on the validity of the templated CDA instances generated in Step 3

```
java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%report_xml_otw% -xsl:%xsl_render% -
o:%report_html_otw%

java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%report_xml_templated% -xsl:%xsl_render% -
o:%report_html_templated%
```

4.4.2.7 Step 6 – Open the HTML validation reports

Variables Used:	*%report_html_otw%, %report_html_templated%*

Opens the HTML validation reports *%report_html_otw%* and *%report_html_templated%*.

```
start %report_html_generic%

start %report_html_templated%
```

4.4.3 The *run_CDAInstanceChecker-Renderer* batch file

The *run_CDAInstanceChecker-Renderer* batch file contains commands to run the following tasks:

- Render the on-the-wire payload instances as HTML

The *run_CDAInstanceChecker-Renderer* batch file utilises the following variables whilst running these tasks:

Variable	Description
%input%	*Directory of on-the-wire CDA instances to be rendered as HTML*
%output %	*Directory where rendered CDA instances will be written following application of the rendering transform*
%xsl %	*XSLT file to use to render the on-the-wire CDA instances as HTML*
%log%	*Log file maintained by the CDA Instance Checker*

The following subsections describe the commands used to run each task within the *run_CDAInstanceChecker-Renderer* batch file. For ease of reading commands to redirect the standard output and error streams to *%log%* have been omitted from these subsections. A full listing of the *run_CDAInstanceChecker-WrappedPayload* batch file is available in Section 5.3.

4.4.3.1 Step 1 – Transform the CDA on-the-wire instances to HTML

Variables Used:	*%input%, %xsl%, %output%*

Utilises Saxon 9.3 library classes to transform the CDA on-the-wire instances at *%input%* to HTML at *%output%* using *%xsl %*.

```
java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar
net.sf.saxon.Transform -s:%input% -xsl:%xsl% -o:%output%

REN %output%\*.xml *.html
```

4.5 Run the CDA Instance Checker

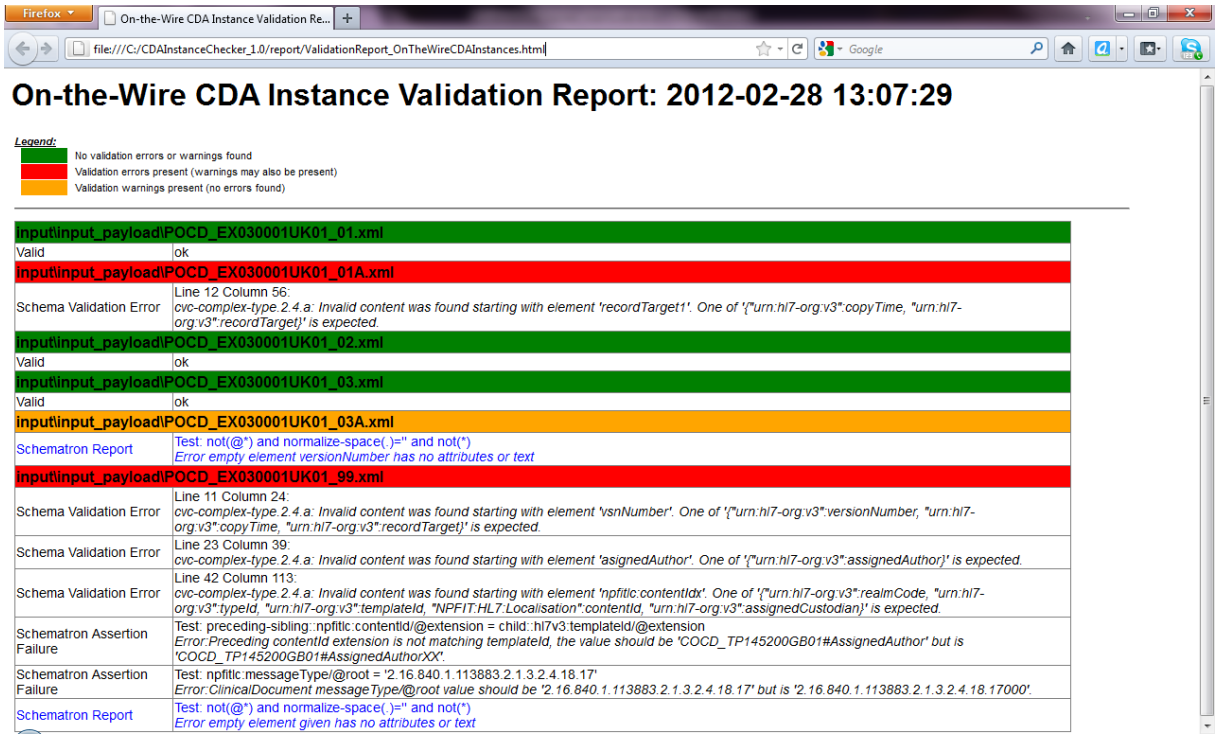
Before running the CDA Instance Checker ensure that:

- The relevant DMS content has been copied as described in section **Error! Reference source not found.**
- The *input\input_payload* or *input/input_wrapped* directory has been populated as described in Section **Error! Reference source not found.**

Then simply execute the relevant batch file in order to run the CDA Instance Checker. Please refer to Section **Error! Reference source not found.** for further details on the batch files.

4.6 View the CDA Instance Checker Validation Report

Following running the *run_CDAInstanceChecker-Payload* or *run_CDAInstanceChecker-WrappedPayload* batch files the HTML validation reports for the on-the-wire and templated CDA instances will be opened automatically. They are stored in the *report* directory should the user wish to re-examine them at a later date.



The CDA Instance Checker validation report uses the following notation:

- Green shaded row – No validation errors or warnings were found in this CDA instance
- Red shaded row – Validation errors were found in this CDA instance. Validation errors may also be present

- Amber shaded row – Validation warnings were found in this CDA instance. No validation errors were found
- **Black** text – is used to provide further information on validation errors that are present
- Blue text – is used to provide further information on validation warnings that are present

4.7 Troubleshooting

4.7.1 Validation errors originating in on-the-wire CDA instances

It should be noted that any validation errors present within the on-the-wire CDA instances used as input for the CDA Instance Checker may provide misleading results within the validation report for the templated CDA instances. If validation errors are reported for the templated CDA instances the user should first ensure that these errors did not originate in the on-the-wire CDA instances before further investigating the templated instance error.

4.7.2 CDA Instance Checker log file

The *CDAInstanceChecker.log* file located within the *log* directory is a useful resource for troubleshooting. All messages reported to the standard output and error stream are redirected to this log file.

5 Appendix

5.1 Appendix A - run_CDAInstanceChecker-Payload.bat

```
REM: ************************************************************
REM: run_CDAInstanceChecker-Payload.bat
REM:   v0.1 - 16-02-2012 - David McGuffin (Applied Informatics Ltd) - Initial draft
REM:   v0.2 - 23-02-2012 - David McGuffin (Applied Informatics Ltd) - Updated with revised naming
convention for directories and variables
REM:   v0.3 - 16-04-2012 - Interoperability Team - Added step to create TemplateLookupId.xml
REM: ************************************************************
REM:
REM:
REM: Set the values of the variables to be used within this batch file.
REM:   Note: Paths can be absolute or relative. Relative paths must be relative to the location of
this batch file.
REM:
REM:   Variables used are:
REM:     - input_mif - directory where all mif files are copied from domain message specification
REM:     - xsl_templatelookup - XSLT file used to create TemplateIdLookup.xml
REM:     - output_templatelookup - directory where TemplateIdLookup.xml file is created.
REM:     - input_payload - directory of on-the-wire CDA instances to be used as input for the CDA
Instance Checker
REM:     - output_templated - directory where templated CDA instances will be written following
transformation
REM:     - schematron_otw - schematron validating stylesheet to be applied to each on-the-wire
instance
REM:     - schematron_templated - schematron validating stylesheet to be applied to each templated
instance
REM:     - xsl_transform - XSLT file to use to transform on-the-wire instance to templated instance
REM:     - xsl_render - XSLT file to use to render the validation report as HTML
REM:     - report_xml_otw - on-the-wire instances XML validation report to be produced by the CDA
Instance Checker
REM:     - report_xml_templated - templated instances XML validation report to be produced by the CDA
Instance Checker
REM:     - report_html_otw - on-the-wire instances HTML validation report to be produced by the CDA
Instance Checker
REM:     - report_html_templated - templated instances HTML validation report to be produced by the
CDA Instance Checker
```

```
REM:     - log - log file maintained by the CDA Instance Checker
REM:     - schema_otw - XML schema to be explicitly applied to each on-the-wire instance. If this
variable has a value of "null" then the CDA Instance Checker will assume that a schemaLocation
attribute is present within the root element of each CDA instance
REM:     - schema_templated - XML schema to be explicitly applied to each templated instance. If this
variable has a value of "null" then the CDA Instance Checker will assume that a schemaLocation
attribute is present within the root element of each CDA instance
REM:

set input_mif=mif
set xsl_templatelookup=transform\createTemplateIdLookup.xsl
set output_templatelookup=schematron
set input_payload=input\input_payload
set output_templated=output\output_templated
set schematron_otw=schematron\Generic_CDA_Document_Schematron.xsl
set schematron_templated=schematron\Templated_CDA_Document_Schematron.xsl
set xsl_transform=transform\TrueCDAToCDALike_v2.xsl
set xsl_render=transform\ValidationReportRenderer.xslt
set report_xml_otw=report\ValidationReport_OnTheWireCDAInstances.xml
set report_xml_templated=report\ValidationReport_TemplatedCDAInstances.xml
set report_html_otw=report\ValidationReport_OnTheWireCDAInstances.html
set report_html_templated=report\ValidationReport_TemplatedCDAInstances.html
set log=log\CDAInstanceChecker.log
set schema_otw=null
set schema_templated=null


REM: Run the CDA Instance Checker. This involves the following processes:
REM:   0. STEP 0 - Transform to create TemplateIDLookup.xml from the available mif files in the domain
REM:   1. Step 1 - Check the on-the-wire instances for validity against the XML Schema and Schematron
REM:   2. Step 2 - Transform the on-the-wire instances to templated instances
REM:   3. Step 3 - Check the newly created templated instances for validity against the XML Schema and
Schematron
REM:   4. Step 4 - Transform the validation reports to HTML
REM:   5. Step 5 - Open the HTML validation reports
REM:
REM: All standard and error stream output is written to %log%
REM:

ECHO. >>%log%
ECHO %date%, %time%: CDA Instance Checker - Payload >>%log%
ECHO ========================================================== >>%log%


ECHO Cleaning up prior to running tool... >>%log%
DEL /Q %output_templated%


ECHO. >>%log%

ECHO STEP 0: Transform to create TemplateIDLookup.xml from the available mif files in the domain
>>%log%
  java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%input_mif% -
xsl:%xsl_templatelookup% -o:%output_templatelookup% 1>>%log% 2>&1


ECHO. >>%log%

ECHO STEP 1: Check the on-the-wire instances for validity against the XML Schema and Schematron
>>%log%
  IF %schema_otw%==null (
    java -classpath lib\xerces-2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar
org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker %input_payload% %report_xml_otw%
%schematron_otw% 1>>%log% 2>&1
  ) ELSE (
    java -classpath lib\xerces-2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar
org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker %input_payload% %report_xml_otw%
%schematron_otw% %schema_otw% 1>>%log% 2>&1
  )


ECHO. >>%log%


ECHO STEP 2: Transform the on-the-wire instances to templated instances >>%log%
  java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%input_payload% -
xsl:%xsl_transform% -o:%output_templated% 1>>%log% 2>&1
```

```
ECHO. >>%log%


ECHO STEP 3: Check the newly created templated instances for validity against the XML Schema and
Schematron >>%log%
  IF %schema_templated%==null (
    java -classpath lib\xerces-2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar
org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker %output_templated% %report_xml_templated%
%schematron_templated% 1>>%log% 2>&1
  ) ELSE (
    java -classpath lib\xerces-2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar
org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker %output_templated% %report_xml_templated%
%schematron_templated% %schema_templated% 1>>%log% 2>&1
  )


ECHO. >>%log%


ECHO STEP 4: Transform the validation reports to HTML >>%log%
  java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%report_xml_otw% -
xsl:%xsl_render% -o:%report_html_otw% 1>>%log% 2>&1
  java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%report_xml_templated% -
xsl:%xsl_render% -o:%report_html_templated% 1>>%log% 2>&1


ECHO. >>%log%


ECHO STEP 5: Open the HTML validation reports >>%log%
  start %report_html_otw% 1>>%log% 2>&1
  start %report_html_templated% 1>>%log% 2>&1
```

5.2 Appendix B - run_CDAInstanceChecker-WrappedPayload.bat

```
REM: ************************************************************
REM: run_CDAInstanceChecker-WrappedPayload.bat
REM:   v0.1 - 27-02-2012 - David McGuffin (Applied Informatics Ltd) - Initial draft
REM:   v0.2 - 16-04-2012 - Interoperability Team - Added step to create TemplateLookupId.xml
REM: ************************************************************
REM:
REM:
REM: Set the values of the variables to be used within this batch file.
REM:   Note: Paths can be absolute or relative. Relative paths must be relative to the location of
this batch file.
REM:
REM:   Variables used are:
REM:     - input_mif - directory where all mif files are copied from domain message specification
REM:     - xsl_templatelookup - XSLT file used to create TemplateIdLookup.xml
REM:     - output_templatelookup - directory where TemplateIdLookup.xml file is created.
REM:     - input_wrapped - directory of wrapped on-the-wire CDA instances to be used as input for the
CDA Instance Checker
REM:     - input_extracted - directory where extracted on-the-wire payloads will be written following
stripping of HL7 wrappers
REM:     - output_templated - directory where templated CDA instances will be written following
transformation
REM:     - schematron_otw - schematron validating stylesheet to be applied to each on-the-wire
instance
REM:     - schematron_templated - schematron validating stylesheet to be applied to each templated
instance
REM:     - xsl_extract - XSLT file to use to extract payload from HL7 wrapped instance
REM:     - xsl_transform - XSLT file to use to transform on-the-wire instance to templated instance
REM:     - xsl_render - XSLT file to use to render the validation report as HTML
REM:     - report_xml_otw - on-the-wire instances XML validation report to be produced by the CDA
Instance Checker
REM:     - report_xml_templated - templated instances XML validation report to be produced by the CDA
Instance Checker
REM:     - report_html_otw - on-the-wire instances HTML validation report to be produced by the CDA
Instance Checker
REM:     - report_html_templated - templated instances HTML validation report to be produced by the
CDA Instance Checker
REM:     - log - log file maintained by the CDA Instance Checker
```

```
REM:     - schema_otw - XML schema to be explicitly applied to each on-the-wire instance. If this
variable has a value of "null" then the CDA Instance Checker will assume that a schemaLocation
attribute is present within the root element of each CDA instance
REM:     - schema_templated - XML schema to be explicitly applied to each templated instance. If this
variable has a value of "null" then the CDA Instance Checker will assume that a schemaLocation
attribute is present within the root element of each CDA instance
REM:

set input_mif=mif
set xsl_templatelookup=transform\createTemplateIdLookup.xsl
set output_templatelookup=schematron
set input_wrapped=input\input_wrapped
set input_extracted=input\input_extracted
set output_templated=output\output_templated
set schematron_otw=schematron\Generic_CDA_Document_Schematron.xsl
set schematron_templated=schematron\Templated_CDA_Document_Schematron.xsl
set xsl_extract=transform\ExtractWrappedPayload.xslt
set xsl_transform=transform\TrueCDAToCDALike_v2.xsl
set xsl_render=transform\ValidationReportRenderer.xslt
set report_xml_otw=report\ValidationReport_OnTheWireCDAInstances.xml
set report_xml_templated=report\ValidationReport_TemplatedCDAInstances.xml
set report_html_otw=report\ValidationReport_OnTheWireCDAInstances.html
set report_html_templated=report\ValidationReport_TemplatedCDAInstances.html
set log=log\CDAInstanceChecker.log
set schema_otw=Schemas\POCD_MT000002UK01.xsd
set schema_templated=null


REM: Run the CDA Instance Checker. This involves the following processes:
REM:   0. STEP 0 - Transform to create TemplateIDLookup.xml from the available mif files in the domain
REM:   1. Step 1 - Extract the on-the-wire payload instances from the HL7 wrapped instances
REM:   2. Step 2 - Check the on-the-wire instances for validity against the XML Schema and Schematron
REM:   3. Step 3 - Transform the on-the-wire instances to templated instances
REM:   4. Step 4 - Check the newly created templated instances for validity against the XML Schema and
Schematron
REM:   5. Step 5 - Transform the validation reports to HTML
REM:   6. Step 6 - Open the HTML validation reports
REM:
REM: All standard and error stream output is written to %log%
REM:

ECHO. >>%log%
ECHO %date%, %time%: CDA Instance Checker - Wrapped Payload >>%log%
ECHO ================================================================ >>%log%

ECHO Cleaning up prior to running tool... >>%log%
DEL /Q %input_extracted%
DEL /Q %output_templated%


ECHO. >>%log%

ECHO STEP 0: Transform to create TemplateIDLookup.xml from the available mif files in the domain
>>%log%
  java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%input_mif% -
xsl:%xsl_templatelookup% -o:%output_templatelookup% 1>>%log% 2>&1

ECHO. >>%log%

ECHO STEP 1: Extract the on-the-wire payloads from the HL7 wrapped instances >>%log%
java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%input_wrapped% -
xsl:%xsl_extract% -o:%input_extracted%  1>>%log% 2>&1


ECHO. >>%log%


ECHO STEP 2: Check the on-the-wire instances for validity against the XML Schema and Schematron
>>%log%
  IF %schema_otw%==null (
    java -classpath lib\xerces-2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar
org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker %input_extracted% %report_xml_otw%
%schematron_otw% 1>>%log% 2>&1
  ) ELSE (
    java -classpath lib\xerces-2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar
org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker %input_extracted% %report_xml_otw%
%schematron_otw% %schema_otw% 1>>%log% 2>&1
  )
```

```
ECHO. >>%log%


ECHO STEP 3: Transform the on-the-wire instances to templated instances >>%log%
  java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%input_extracted% -
xsl:%xsl_transform% -o:%output_templated% 1>>%log% 2>&1


ECHO. >>%log%


ECHO STEP 4: Check the newly created templated instances for validity against the XML Schema and
Schematron >>%log%
  IF %schema_templated%==null (
    java -classpath lib\xerces-2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar
org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker %output_templated% %report_xml_templated%
%schematron_templated% 1>>%log% 2>&1
  ) ELSE (
    java -classpath lib\xerces-2_9_0\xercesImpl.jar;lib\cdaInstanceChecker-1_0\cdaInstanceChecker.jar
org.net.nhs.hl7.cdaInstanceChecker.CDAInstanceChecker %output_templated% %report_xml_templated%
%schematron_templated% %schema_templated% 1>>%log% 2>&1
  )


ECHO. >>%log%


ECHO STEP 5: Transform the validation reports to HTML >>%log%
  java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%report_xml_otw% -
xsl:%xsl_render% -o:%report_html_otw% 1>>%log% 2>&1
  java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%report_xml_templated% -
xsl:%xsl_render% -o:%report_html_templated% 1>>%log% 2>&1


ECHO. >>%log%


ECHO STEP 6: Open the HTML validation reports >>%log%
  start %report_html_otw% 1>>%log% 2>&1
  start %report_html_templated% 1>>%log% 2>&1
```

5.3 Appendix C - run_CDAInstanceChecker-Renderer.bat

```
REM: ************************************************************

REM: run_CDAInstanceChecker-Renderer.bat

REM: v0.1 - 27-02-2012 - David McGuffin (Applied Informatics Ltd) - Initial draft

REM: v0.2 - 05-04-2013 - Interoperability Team

REM: ************************************************************

REM: Set the values of the variables to be used within this batch file.

REM:   Note: Paths can be absolute or relative. Relative paths must be relative to the location of
this batch file.

REM:   Variables used are:

REM:     - input - directory of on-the-wire CDA instances to be rendered as HTML

REM:     - output - directory where rendered CDA instances will be written following application of
the rendering tansform

REM:     - xsl - XSLT file to use to render the on-the-wire CDA instances as HTML

REM:     - log - log file maintained by the CDA Instance Checker

set input=input\input_payload

set output=output\output_rendered

set xsl=transform\nhs_CDA_Document_Renderer.xsl

set log=log\CDAInstanceChecker.log
```

```
REM: Run the CDA Instance Checker Renderer. This involves the following processes:

REM:   1. Step 1 - Render the on-the-wire payload instances as HTML

REM: All standard and error stream output is written to %log%

ECHO. >>%log%

ECHO %date%, %time%: CDA Instance Checker - Renderer >>%log%

ECHO ============================================================ >>%log%

ECHO Cleaning up prior to running tool... >>%log%

DEL /Q %output%

ECHO. >>%log%

ECHO STEP 1: Render the on-the-wire CDA instances as HTML >>%log%

java -classpath lib\saxonhe9-3-0-5j\saxon9he.jar net.sf.saxon.Transform -s:%input% -xsl:%xsl% -
o:%output%  1>>%log% 2>&1

REN %output%\*.xml *.html 1>>%log% 2>&1
```