Province of British Columbia

Guidelines to the Indigenous Languages Technology Standard

Version 1.1

January 21, 2025

Table of Contents

Purpose

These guidelines outline how to support Indigenous languages in computer systems, complementing the material in the Indigenous Languages Technology Standard.

Description

Action 3.15 of the Declaration Act Action Plan commits the Province to "adopt an inclusive digital font that allows for Indigenous languages to be included in communication, signage, services and official records." To achieve this, the Province's IM/IT systems need to meet specific technical capabilities.

These guidelines help ministries to ensure IM/IT systems can support Indigenous languages to deliver more inclusive government services to Indigenous people living in B.C.

By supporting Indigenous languages in systems and services, government will be able to:

- Record Indigenous-language names for people, places, or businesses.
- Display Indigenous-language names in provincial applications, data, and mapping systems.
- Print signs, correspondence, and documents that contain Indigenous languages.

Application

All entities (hereafter, "ministries") identified in Core Policy and Procedures Manual Chapter 1, section 1.2.4.

Authority

Core Policy and Procedures Manual Chapter 12

Advice on these guidelines

For questions or comments regarding these guidelines, please contact:

BC Data Service, Ministry of Citizens' Services

How to support Indigenous languages in IM/IT systems

An important step to including Indigenous languages in government's records, systems, and services is ensuring government's technology systems can support them.

This guide describes how to assess and update IM/IT systems so they can support Indigenous languages.

Assess your systems for Unicode readiness

We suggest a four-step approach to assess your systems for Unicode-readiness:

1. Learn the foundational IM/IT terminology and concepts.
2. Review the system components.
3. Identify potentially problematic string operations.
4. Test data flows and evaluate data exchanges for Unicode-readiness.

These are described in the following sections.

Learn the foundational IM/IT terminology and concepts

Review important terminology and the history of language to make it easier to complete your Unicode-readiness assessment.

Understanding character sets and encodings

A character set is a list of characters supported by a computer system, and an encoding scheme describes how to store them on a computer system as ones and zeroes (binary data). Character sets vary in the number of characters in the set (the size of the *repertoire*). Encoding schemes vary in the number of bytes required to store a character and whether this number varies.

Older character sets

Many older B.C. government IM/IT systems support just the characters available on the US ASCII keyboard. Many systems are based on the z/OS® (mainframe) operating system; these use a different character set/encoding scheme called EBCDIC. Some systems support extended forms of ASCII: Windows-1252 and ISO-8859-1. These can store accented characters found in North European languages. All these character sets have encodings which require just one byte per character. Because of their limited size, they cannot contain all characters used in Indigenous languages in B.C. Unicode is the only character set large enough to support the Indigenous languages in B.C.

Unicode

Unicode is an international encoding standard created in the early 1990's. Its goal was to produce a single, unified standard that supports all the characters used in any of the world's living languages. Unicode contains over one million characters, including those used by B.C. Indigenous languages.

The Unicode set is updated every year as new characters are added. For example, in the 2024 release (Unicode Version 16), three characters used in the B.C. west coast Haíłzaqv language were added.

To process Unicode data, all the system's data stores need to be configured to store data in Unicode's standard encodings. In B.C. government systems, the standard encoding is UTF-8. UTF-8 is a varying-length encoding, requiring between 1 and 4 bytes of storage per character. An advantage of UTF-8 is that all the characters in the ASCII repertoire take just 1 byte each when stored.

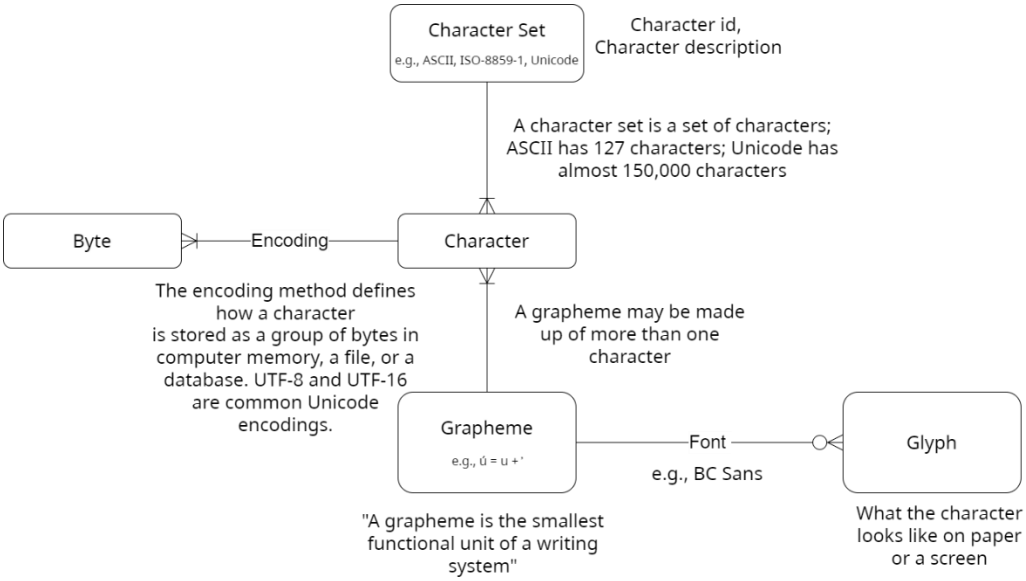
The Unicode character set enables systems to use B.C.'s inclusive font (BC Sans). BC Sans can display text from multiple languages including every character used in Indigenous languages in B.C.

Modernizing: converting to Unicode

Many B.C. government systems have not undergone modernization, due to their complexity and the risk of service delivery issues like data loss, corruption or security risks. For example, there may be parts of the code that assume characters always use just one byte of storage; these programs will break when they encounter variable-length, multi-byte characters.

Understanding language processing

There are many things that impact how IM/IT systems process languages. The following is a data architecture entity relationship diagram showing how terms like "byte", "font", "encoding", "grapheme", "glyph" and "character set" relate to one another:



At the bottom of the diagram, note that what we see as a single character could actually be many characters superimposed upon one another creating a grapheme. For example, the character "c cédille" combines the Latin character 'c' with a superimposed cedilla accent.

How a grapheme appears when displayed on the screen or paper is determined by the font used. BC Sans, for example, is a font influencing the visual representation of graphemes, referred to as glyphs.

Review the system components

The next step is to identify the technologies used in your systems to create a gap assessment for Unicode-readiness. Systems will vary in how “unicode ready” they are and what needs to be done to make the “unicode ready.”

Table 1: Characteristics of Unicode-ready systems

Characteristic	**Likely Unicode-Ready**	**NOT likely Unicode-Ready**
Programming language	Java version18 ; Python 3.x; JavaScript	C; C++; Python 2; PHP
Database encoding	UTF-8; UTF-16; UTF-32	iso-8859-1(Latin1); windows-1252 (Western European); ASCII

Characteristic	Likely Unicode-Ready	NOT likely Unicode-Ready
String handling (system queries, searching, sorting, etc.)	Strings treated as sequence of bytes, not individual characters. Extra space allocated for string variables and database columns.	Strings treated as sequences of characters. Possibly needing to know length of string in characters. What is the *n*th character, etc. No extra space allocated for string variables and database columns.
Data edit rules	Program logic does not restrict inputs to a specific set or range of characters.	Program logic restricts inputs to a specific set or range of characters.
Web server and associated files	Web server configuration file (e.g., httpd.conf) has a directive recognizing Unicode, or individual pages have a UTF-8 meta tag.	Web server configuration file (e.g., httpd.conf) does not have a directive recognizing Unicode, and individual pages do not have a UTF-8 meta tag.

Databases and Unicode

When building a database, the character set and encoding combination must be specified for data storage. How this encoding is identified differs by database. The following table lists the acceptable Unicode character set encodings for several popular database management systems. These are all UTF-8 encodings.

UTF-8 is the preferred encoding for Unicode data as it is very space efficient – with UTF-8, ASCII characters (letters, numbers, many punctuation characters) use just one byte each to store. A Unicode database that contains just ASCII characters will consume the same amount of space as an ASCII database.

Table 2: Preferred encodings for popular databases

Database	Unicode encoding
Oracle	AL32UTF8
PostgreSQL	UTF8 (character_repertoire = UCS)
SQL Server	ends in '_utf8'
MySQL	utf8mb4
MongoDB	UTF8 (This is standard for MongoDB)

Code samples and instructions on how to determine and set the encoding for a database are available in DevHub.

Collation rules

Collation rules determine how characters are sorted, and strings are compared, impacting factors like case sensitivity, accent relevance and character arrangement in sorting.

Different languages often have specific collations; for instance, French and English differ as described in the Oracle Database Globalization and Support Guide. Binary comparison and sorting, which rely on binary encodings, are frequently used. However, binary comparison and sorting does not work well where there is a mixture of Latin (A-Z, 0-9) and non-Latin Unicode data, as in B.C. Indigenous languages. For this reason, Unicode provides a Unicode collation algorithm that can be incorporated into database systems when setting system parameters.

The following table lists the preferred collation setting for some common database management systems:

Table 3: Preferred collation rules for popular database management systems

Database	**Collation**
Oracle	UCA0700_ORADUCET
PostgreSQL	unicode
SQL Server	ends in '_utf8'
MySQL	starts with 'utf8mb4_' (e.g., utf8mb4_0900_ai_ci)
MongoDB	UTF8 (This is standard for MongoDB)

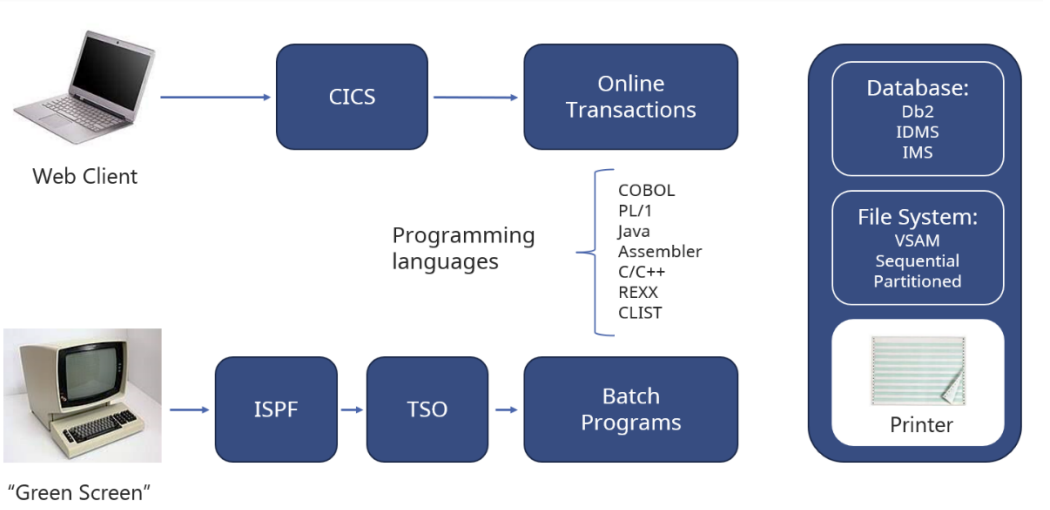
Further details are available in DevHub

Keyboards

Enabling the data search function in an application involves more than just setting the collation rules correctly. You must also provide a way for the person using the application to specify the text to be searched. This is more complicated when the text includes characters not found on the US ASCII keyboard. The First Voices program provides keyboards for enabling search, and smart phone apps for Apple and Android.

Mainframe systems

Mainframe systems deserve special coverage, as their components may or may not support Unicode. The following diagram illustrates some of the components present in mainframe systems:



Some of these components were designed around a specific non-Unicode character set (EBCDIC) and do not work with any other character set. These include:

- Green screen terminals (IBM 3270 terminals and emulators)
- Mainframe printers

The other mainframe components shown in the diagram, while originally designed for use with EBCDIC, can now support Unicode. See DevHub for more information.

Web servers

Web servers can be configured to serve web pages that have Unicode content.

In Apache web servers, this configuration can be done by adding the following line to the httpd.conf configuration file:

```
'AddDefaultCharset utf-8'
```

When configured this way, a page with the following html will appear, correctly, as Tk'emlúps te Secwépemc.

```
<html><body>
<h1>Tk'emlúps te Secwe'pemc </h1>
</body></html>
```

If that line is missing from the configuration file, then the same page might appear like Tkâ€™emlÃºps te SecweÌ“pemc.

Regardless of how the web server is configured, Unicode support can be ensured by including a directive in the web page itself. For example, the following page will render properly even if the web server has not been configured for Unicode support, since the web page includes a meta charset directive:

```
<html><body>
<meta charset="UTF-8">
<h1>Tk'emlúps te Secwe'pemc </h1>
</meta>
</body></html>
```

File format

Data stored in files must be encoded in UTF-8. Specific guidance depends on the format of the file:

i. CSV: Comma-separated value (CSV) files must include a byte order mark (BOM) at the start of the file to render properly in Excel.

ii. DBF: The dBase File Format (DBF) is a file format used by older desktop databases such as dBase, Clipper, and FoxPro. It is also used to store non-spatial data in Esri shapefiles. Originally designed to store ASCII data, in more recent versions the default encoding was changed to ISO-8859-1. DBF files can store Unicode data encoded as UTF-8, but this is not the default.

iii. PDF: UTF-8 / BC Sans Microsoft Office documents converted to PDF through Microsoft Office will retain their UTF-8 encoding, and the BC Sans font will be embedded, allowing users to view the document even if they do not have BC Sans installed on their system. No extra action is required.

iv. Microsoft Office: Unicode UTF-8 is the default encoding used in Microsoft Office products (e.g., Word, Excel, PowerPoint); no extra action is required.

See DevHub for more information on how to make sure data files can properly support Indigenous language characters.

Identify potentially problematic string operations

Once you have ensured your databases, programming languages and web servers can support Unicode, the next step is to identify potentially problematic system (string) operations.

Computer programs will need to be modified to support Unicode text if they perform operations like:

- Sorting
- Searching
- Matching

Programs written to support strings with byte-sized constituent characters will produce errors when dealing with multi-byte, varying-length UTF-8 encoded Unicode characters.

As described earlier, what might look like single characters in Indigenous languages are sometimes compositions of multiple Unicode characters. For example, the symbol é is made up of two Unicode characters:

- The Latin letter e
- An overlaid acute accent.

These compositions are called graphemes. To support the processing of Indigenous language text, programs must be able to segment the text into graphemes. No common programming language has built-in support for graphemes. Libraries for doing this are available in most common programming languages.

Some Unicode characters with diacritics can have multiple UTF-8 encodings, making it challenging to search for them. To address this issue, Unicode provides normalization forms that can remove ambiguity during searching.

DevHub contains code samples illustrating how to work with Unicode characters and graphemes in several common programming language.

String manipulation

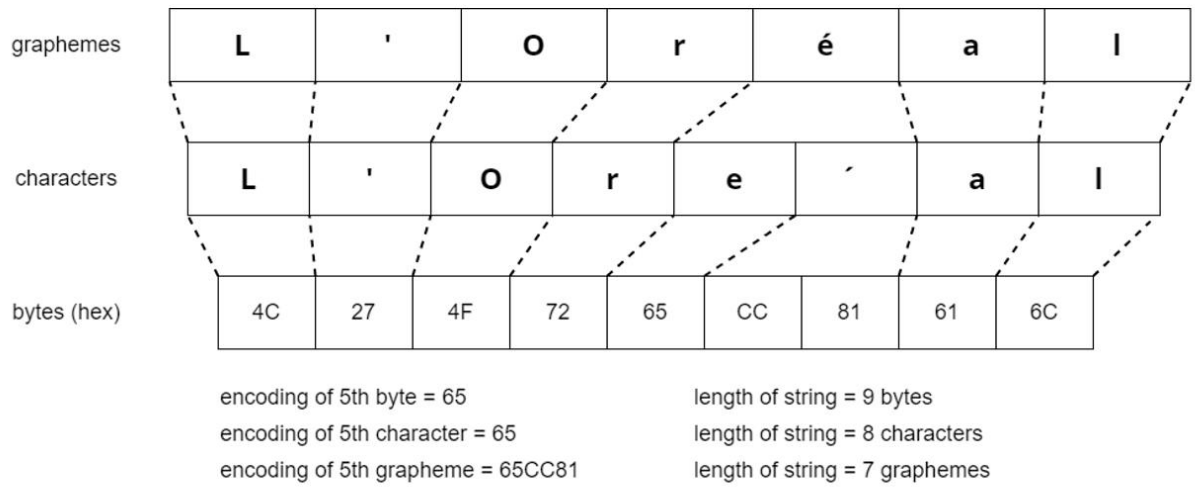
There are several common operations that can be applied to text strings. These may behave unexpectedly when applied to text strings that contain Unicode characters.

String length

With Unicode data, the number of characters can be less than the number of bytes required for the encoding, which can be up to 4 bytes per character. The functionality you are programming may need you to know the length of a text string in terms of the number of:

- Bytes: to determine the amount of storage required when storage is allocated by number of bytes
- Characters: to determine the amount of storage required when storage is allocated by number of characters
- Graphemes: to allocate screen space

Some graphemes, such as the “e” combined with an acute accent (‘é’) are made up of multiple characters. In such a case, the number of graphemes in a string is less than the number of characters, and the number of characters is less than the number of bytes.



String comparison

With non-ASCII Unicode text strings, checking that two text strings are equal is complicated because two logically identical strings might have different encodings, as described above. The logical order of two text strings is also not always clear.

Position in string

For ASCII data, the position of a particular character in a text string does not depend on whether the measurement is bytes or characters. The n^{th} byte and the encoding of the n^{th} character in an ASCII text string are the same.

For non-ASCII Unicode data, this is not the case. In the image above, the position of 'a' is the 6^{th} grapheme, the 7^{th} character, and the 8^{th} byte.

Substring

The "substring" operation can be viewed as positioning in a text string, then extracting a specific length of data. The complexities of performing the substring operation on non-ASCII Unicode data are a combination of those in string length and position in string (see above).

Encryption and decryption

There are various methods for encrypting data before storing or transmitting it and then decrypting it upon retrieval. Encryption and decryption methods that work well when the subject data can be viewed as a string of fixed-size, single-byte characters may not work when the characters have multi-byte or varying length encodings as non-ASCII Unicode characters do.

The Cryptography with International Character Sets guidance provides two principles to keep in mind when encrypting/decrypting non-ASCII Unicode data:

1. Work with bytes, not text strings
2. Do not store encrypted data in a string type

Different encoding methods with Unicode can produce different binary representations of the same text. Hence, any system decrypting data from another system must know the encoding used in the original system.

Test dataflows and evaluate data exchanges for Unicode readiness

Once you have ensured your string operations can support Unicode, the final step is to test your dataflows for Unicode-readiness.

Examine how data moves through your system, from entry, processing, storage, and output. Identify the other systems your system communicates with and check if they can use Indigenous languages.

The next step provides guidelines on how to test and evaluate data exchanges for Unicode-readiness. This is the last stage of how to assess your systems for Unicode-readiness.

Investigate system data flow

The final task to confirm if your system is ready for Unicode is to investigate how a text string that contains Unicode characters flows into, through, and beyond the system.

Consider a simple system that:

1. Inputs a name
2. Stores it in a database,
3. Does a query on the name to find related information,
4. Outputs this related information.

You can test this system by inputting a name containing Unicode characters, then checking that the output is as expected and that no errors are generated.

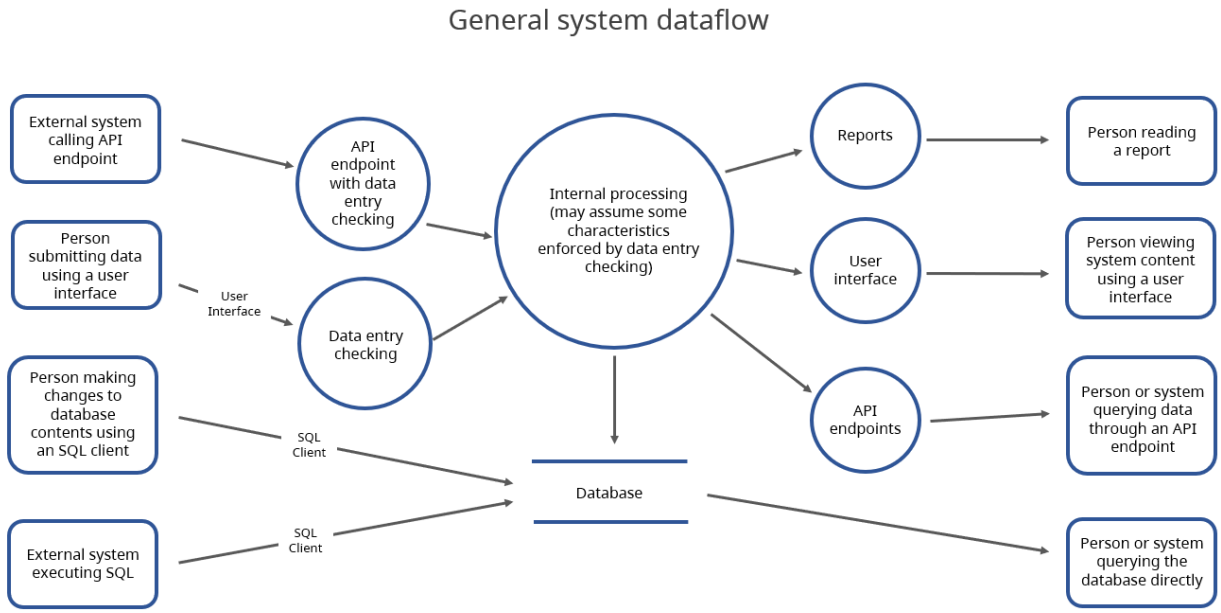
Dataflow diagrams

Dataflow diagrams can be used to model the flow of data:

- Into and out of the system
- Into and out of the processes within the system
- Into and out of data stores such as files and databases

To access data flow diagrams or resources and standards for the creation of your diagrams within B.C. government, contact the area within your organization responsible for IM/IT system management (sometimes called the Information Management Branch, Computing Services Branch or similar).

For a typical system that inputs and outputs “name” data, the data flow diagram might look like this:



The system might guard against invalid input being entered, then do some processing on the accepted data. It may store the data in a query-able database and/or make the data available to consumers through an API. A dataflow diagram captures all these touchpoints when data needs to be handled in a way that supports Indigenous languages.

Dataflows are useful in assessing whether a system will properly handle specific types of data. View example dataflows.

Dataflow architecture

Dataflow architecture, typically expressed in data flow diagrams, have four types of objects:

- Entities
- Internal processes
- Data stores
- Dataflows

Entities (Actors)

Entities, also known as actors, are the users or processes that input or output data to or from a system. In a dataflow diagram these are indicated using a rectangle that includes text describing what the user or process is doing.

To assess Unicode-readiness, you must identify all the ways that strings containing Unicode characters can enter and leave the system. This includes checking Unicode characters display correctly on output entities like:

- PDF documents
- Screen outputs
- Printed reports

Review test data and guidelines that can be used to test your systems for Indigenous language capabilities.

Processes

A process is a sequence of actions performed on a data element as it moves from its input to its destination. The end point determines whether the data will leave the system or be stored in it. Data flow diagrams represent processes with circles.

To assess Unicode-readiness you must identify all the processes that can operate on strings containing Unicode characters.

Data stores

Data stores are the places where data gets stored within a system. Data stores include:

- Files in a file system
- Tables in a database

Data stores are represented by parallel horizontal lines (a box with no sides).

Dataflows

Dataflows connect entities with processes and data stores. They are represented by directed lines. In assessing Unicode-readiness, each dataflow needs to be tested.

File formats and transfer types

Where applicable, include notes to identify the types of files being read or written (e.g., PDF, CSV, Excel, etc.), and any file transfer protocols used (e.g., FTP, HTTPS, SFTP etc.)

Evaluate data exchanges

Assess compatibility of other systems that exchange data with yours. If these systems are not compatible with Unicode, take measures to ensure that data is exchanged correctly.

- Refer to W3C's International Workgroup's Creating a Roadmap for guidance

Assessment is the only way to know whether an existing system is Unicode-ready. Assessing your systems will also help you understand which parts, if any, are problematic.

Update or replace your system

Depending on the results of your assessment, your system may need a simple fix to support Indigenous languages, or it may need much larger upgrades.

Developer resources

We have set up a DevHub repository to assist in adapting existing systems or creating new ones compatible with Unicode. The site contains resources to support assessment, testing and other developer activities. Resources include:

- Test data
- Mapping of Unicode encodings
- Examples of data flow analysis
- Code samples and much more

Funding upgrades

If your system requires IM/IT capital funding to align with the ILTS and these guidelines, you should review the Digital Investment Office's Digital Investment 101. Digital Investment 101 will help you plan and submit requests for capital IM/IT funding.

Procurement support

If you are considering procuring IM/IT services and need support in ensuring vendors are aware of the requirement to align with the ILTS please contact us.

Revision history

Version	Date	Notes
1.0	July 2024	Web version
1.1	January 2025	PDF version