Australian Bureau of Statistics

ABS Occupation Coding Service

User Guide
(API integration guide)

June 2025

Australian Bureau of Statistics

Contents

1. Background

1.1 Introduction

The Whole-of-Australian-Government (WoAG) Occupation Coding Service ('the Coding Service' or 'the service') has been designed by the Australian Bureau of Statistics (ABS) to provide a single occupation coder across government, industry and the community.

The Coding Service will code occupation data to the latest Australian standard occupation classification titles and codes.

1.2 Design

The Coding Service has been built with supervised machine-learning technology, to train hierarchical support vector machine (HSVM) models that provide high-quality and comprehensive automated coding against hierarchical classification categories. A Confidence Threshold is applied to the service, ensuring that outputs are high quality.

The service is called via an Application Programming Interface (API), designed to support integration across systems and platforms (including online forms and survey instruments) by offering authenticated, standards-compliant endpoints.

All API services are hosted in Australia to comply with relevant data sovereignty and privacy regulations.

Users have the option to register as a public user (throttled service), or a partner user (enhanced capability). Public and partner registrations enable the following services:

Public user	- Single record (synchronous) coding - Small batch synchronous coding (up to 300 records)
Partner user	- Single record (synchronous) coding - Small batch synchronous coding (up to 300 records) - Large-file asynchronous upload/download bulk coding (from 1 record to millions of records)

Public user real time coding API calls will be throttled at 1 request per second, within a ceiling of 100 requests per hour.

1.3 Security and Technology Standards

The WoAG Coding Service and API has been security assessed by an independent registered assessor within the Australian Signals Directorate (ASD) Information Security Registered Assessors Program (IRAP) Program. This assessment found the WoAG Coding Service and API to have met the control and security objectives defined through the Australian Government Information Security Manual (ISM).

The service has been built to comply to the Australian Information Security Manual (ISM) and the Protective Security Policy Framework (PSPF). It leverages modern web API technologies in accordance with the Australian Government's API Standard and globally recognised security

frameworks. These standards ensure that the service is designed for safe, scalable integration across government and public-facing systems.

Both public and partner service users will be registered, and will be provided with relevant authorisation tokens to access the service.

See Section 12, Security, for more detail, including security controls to assist partner agencies in assessing their risks when using this service.

1.4 Using the guide

This User Guide supports access to and use of the Occupation Coding Service. It is **targeted toward software developers and technical professionals integrating the service into a client application**.

It outlines the API endpoints available for accessing and using the service, and provides integration instructions for calling the API.

The guide is structured to be followed sequentially from Section 2 (Getting started) through to Section 6 (Gathering parameters).

Users will then proceed to Section 7 (Synchronous, or real-time coding) for single record or small batch coding, or Section 8 (Asynchronous batch coding), depending on the data to be coded.

1.4.1 Synchronous Coding

The synchronous single-record coding service is designed for real-time usage (~1 second per record). It is suitable for small volumes and live systems such as online forms, web surveys, or ABS site tools.

Synchronous coding also supports coding small batches of records (up to 300 records) with similar per-record timing.

Note: This service is not optimised for large volumes and should not be used for high-throughput workloads. Use asynchronous coding for scalable batch processing.

1.4.2 Asynchronous Coding

Asynchronous batch coding is designed for large datasets (from a single record to millions of records). While asynchronous coding is the most efficient service for larger batches of data, it is not real-time, and may be queued during high load periods.

Batch uploads are submitted via the API, and status is checked via polling (operation endpoints). Response times for batch requests may range from a few minutes to several hours depending on file size, system demand and current queue load at the time of submission.

1.5 Errors and Glossary

- A list of errors and suggested actions is provided at Section 10.
- A glossary of input types and response formats is provided at Section 11.

2. Getting Started

2.1 Pre-testing readiness

Before accessing the service, you will need to register for the coding service (see Section 2.3 below). You will also need to consider the following:

- What you need to set up to pass the API packet to your API endpoint (url).
- What data you are going to code.
- Whether you need single record coding, small batch coding (up to 300 records), or large batch coding.
- Whether you need to clean the data first (for example, batch data will need to fit specific formats).

See also Section 4, Coding Service formats.

2.2 Terms of Use

The use of the Coding Service API is governed by the Coding Service Terms of Use. All API users will be required to accept these Terms of Use prior to gaining access to the service.

Users requesting access to the service must be appropriately authorised to accept the Terms of Use on behalf of their organisation (link to webpage)

2.3 Registration

To register for the service and request client credentials, please complete the following request form: (link to webpage)

Once you have registered, your ABS contact will email you:

- A client identifier, `clientID`.
- A client secret, `clientSecret`. This must be kept confidential as it is used when authenticating your requests.

The ABS will monitor registrations for usage, and users may be notified via email if their access is under review for removal due to inactivity.

3. Authentication

An authentication token is required to use the Coding Service. This is a unique, time-limited access key which is used to authenticate all API calls to the service.

3.1 Get an authentication token

To get an authentication token, you must first call the service's authentication endpoint with an authorisation header. More details are available in the AWS documentation, but the key input parameters are described below:

3.1.1 Request Syntax

```
POST /oauth2/token HTTP/1.1
Host: https://partner-coder.auth.abs.gov.au
Content-Type: application/x-www-form-urlencoded
Authorisation: string
{
    grant_type: "client_credentials"
}
```

3.1.2 Request Header Parameters

Authorisation

Basic authorisation method with a base64 authorisation token (`encodedAuthString`), computed from the client ID and client secret provided upon registration.

`encodedAuthString` can be computed via the bash command:

```
$ echo -n "${clientID}:${clientSecret}" | base64
```

Type: String

3.1.3 Response Syntax

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    access_token: "string"
}
```

3.1.4 Response Elements

access_token

Your unique access token which can be used to authenticate all API calls to the coding service.

Type: String

3.1.5 Examples

On registering for the coding service, this user was issued with the following:

- `ClientID: “client1”`
- `ClientSecret: “secret123”`

`encodedAuthString` should be the base64 encoding of “`client1:secret123`” and the entire request is as follows:

Sample Request

```
POST /oauth2/token HTTP/1.1
Host: https://partner-coder.auth.abs.gov.au
Content-Type: application/x-www-form-urlencoded
Authorisation: Basic Y2xpZW50MTpzZWNyZXQxMjM=
{
    grant_type: "client_credentials"
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    access_token: "example token"
}
```

3.2 Use an authentication token

To authenticate against the coding API service, you will need to include your access token in the header of any API calls.

Your token will last one hour from the time of issue, after which you will need to request a new token.

The API calls are of a short duration, usually less than a few seconds. When initiated, each call will check the authentication and then continue with the rest of the call. If the call was approved at the start, it will return a response if the timer runs out.

Asynchronous batch calls may take longer to return results, and you may have to re-authorise to receive the results.

3.2.1 Request Header Parameters

Authorisation

The authorisation token retrieved via the Authentication mechanism.
Type: String

4. Coding service formats

4.1 Request formats

The coding service uses JSON format for the following services:

- Real-time (synchronous) public coding service
- Real-time partner coding service
- Real-time small batch coding service

It uses JSONL format for the large batch/bulk (asynchronous) partner coding service.

The GET Data method will return the following for each of the specified services for occupation coding:

Service	Returns
Real-time (synchronous) public coding service	• One or more classification codes and titles for the free text supplied
Real-time partner coding service	• One or more classification codes and titles for the free text supplied
Real-time small batch coding service (up to 300 records)	• The best match 1-digit to 6-digit codes and titles (moving up the classification hierarchy from 6-digit to 1-digit level) for the free text supplied • If the coder cannot code the free text supplied, it will provide 3 suggestions
Large batch/bulk (asynchronous) partner coding service	• The best match 1-digit to 6-digit codes and titles (moving up the classification hierarchy from 6-digit to 1-digit level) for the free text supplied • If the coder cannot code the free text supplied, it will provide 3 suggestions

4.2 Recommended text input for coding

The occupation coder will perform optimally when provided with both a job title and tasks as free text inputs, as this is how the ML training was carried out. The coder will not perform as well with just one text field entered (i.e if only the job title or only the task text is entered). If results are unsuccessful, entering more information will help the Coding Service make better predictions.

Text strings can be a **maximum of 100 characters only** (a total of 100 characters for combined occupation and task input text entries).

The coding service API will not accept custom data queries or query string parameters.

The occupation coding models are trained and optimised for use in an Australian context. They use an English character set for coding responses that relate to individuals employed in legal occupations that fall within the definition and scope of official ABS occupation classifications. For example, ‘retiree’ and ‘homemaker’ do not return codes, as these are not occupations defined in official ABS occupation classifications.

The coding service has been trained on English inputs only. The service accepts printable ASCII characters, which includes all English letters and connectives, but excludes certain accents, foreign currency symbols and control characters like file endings or backspace. Including a bad character may result in an `Invalid request body` error.

The contextual assumption of the input text is that the text relates to and describes a person's job. The coder, being able to recognise a very broad vocabulary, will attempt to code all input text, regardless of context. Therefore, users need to ensure a contextual fit between their input data and the coding task being undertaken by the coder.

For instance, if a person describes their job as a 'prisoner', the Coding Service assumes a context that the occupation to be coded works with prisoners in some way, and codes to 'Correctional Officer'. Likewise, the input text 'baby' codes to 'Nanny'.

4.3 Multiple occupation entries

The service is designed to provide a *single* occupation code and title for a *single* occupation record. If multiple occupations per record are entered in the occupation title text input, the coder will attempt to code the provided text to a best fit single occupation code at the most detailed level.

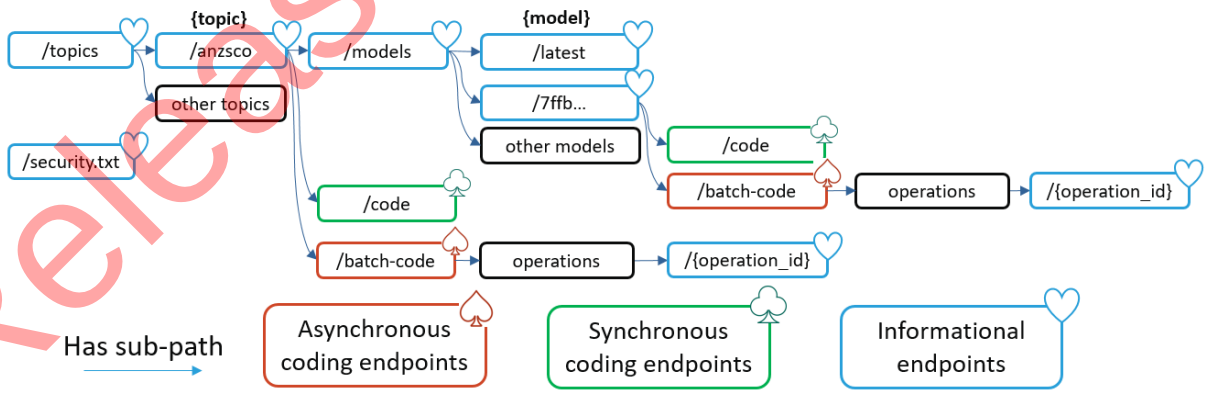
The output will reflect the training data and will depend on how many times the two jobs were present together in the training data. The Coding Service will default to whatever is most commonly found in the training data.

If multiple occupations are present, you will need to format each job as a separate request.

5. API Endpoints and HTTP methods

The API endpoints and their HTTP methods are outlined below in both the table and the diagram.

Endpoint	HTTP verb
/v1/topics Retrieve a list of available topics.	GET
/v1/topics/{topic} Describes the input format for the given topic.	GET
/v1/topics/{topic}/code Synchronously codes a single record or small batch of records against the latest model for a given topic.	POST
/v1/topics/{topic}/models Lists the available models and their input formats for a given topic.	GET
/v1/topics/{topic}/models/latest Describes the input format for the latest model for a given topic.	GET
/v1/topics/{topic}/models/{model}/code Synchronously codes a single record or small batch of records against a specific model for the given topic.	POST
/v1/topics/{topic}/batch-code Creates a new asynchronous batch inference operation against the latest model for a given topic.	POST
/v1/topics/{topic}/models/{model}/batch-code Creates a new asynchronous batch inference operation against a specific model for the given topic.	POST
/v1/topics/{topic}/batch-code/operations/{operation_id} Checks the status of a batch inference operation.	GET
/v1/topics/{topic}/models/{model}/batch-code/operations/{operation_id} Checks the status of a batch inference operation.	GET
/v1/security.txt Returns contact details for reporting issues.	GET



6. Gathering Parameters

6.1 Listing available topics

Before coding against a topic (classification), you must confirm that the topic is supported by the application. You will also need to record its corresponding `uriName` for further calls to the API.

6.1.1 Request Syntax

```
GET /v1/topics HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: string
```

6.1.2 URI Request Parameters

The request does not use any URI parameters.

6.1.3 Request Body

The request does not have a request body.

6.1.4 Response Syntax

```
HTTP/1.1 200 OK
Content-type: application/json
[
    {
        "uriName": "string",
        "fullName": "string"
    }
]
```

6.1.5 Response Elements

If the action is successful, the service sends back an HTTP 200 response. The API returns an array of Topic objects representing all the coding topics currently supported by the API.

6.1.6 Errors

For information about the errors that are common to all actions, see Section 10, Errors and suggested actions.

6.1.7 Examples

Sample Request

```
GET /v1/topics HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
```

```
Content-type: application/json
Authorisation: example token
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json
[
    {
        "uriName": "osca",
        "fullName": "OSCA - Occupation Standard Classification for Australia"
    }
]
```

6.2 Getting the input format for the latest model for a topic

Different models are coded against different input formats. If you are using the latest (default) model for your specified topic, you can get the input format via the following mechanism.

If you are using another model, the input format will be provided as part of the list of available models.

6.2.1 Request Syntax

```
GET /v1/topics/{topic}/models/latest HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: string
```

6.2.2 URI Request Parameters

topic
The `uriName` for the coding topic of interest. This can be acquired by listing the available topics.
Required: Yes

6.2.3 Request Body

The request does not have a request body.

6.2.4 Response Syntax

```
HTTP/1.1 200 OK
Content-type: application/json
{
    "modelId": "string",
    "modelVersion": number,
    "modelReleaseDate": "string",
    "modelType": "string",
    "inputFormat": [
```

```
        "string"
    ],
    "topicStandard": "string",
    "topicVersion": "string"
}
```

6.2.5 Response Elements

If the action is successful, the service sends back an HTTP 200 response. The API returns a Model object representing the latest model for the given topic, which includes the expected input format.

6.2.6 Errors

For information about the errors that are common to all actions, see Section 10, Errors and suggested actions.

6.2.7 Example

Getting the latest occupation model

Sample Request

```
GET /v1/topics/osca/models/latest HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
    "modelId": "00000000-0000-0000-0000-000000000000",
    "modelVersion": 2,
    "modelReleaseDate": "2023-12-20T06:06:44.514Z",
    "modelType": "hsvm2",
    "inputFormat": [
        "occp_text",
        "tasks_text"
    ],
    "topicStandard": "OSCA - Occupation Standard Classification for Australia",
    "topicVersion": "2024"
}
```

6.3 Listing available models for a given topic

To code against a specific machine learning model, you can browse the available models and their input formats by calling this endpoint.

6.3.1 Request Syntax

```
GET /v1/topics/{topic}/models HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: string
```

6.3.2 URI Request Parameters

topic
The `uriName` for the coding topic of interest. This can be acquired by listing the available topics.
Required: Yes

6.3.3 Request Body

The request does not have a request body.

6.3.4 Response Syntax

```
HTTP/1.1 200 OK
Content-type: application/json
[
    {
        "modelId": "string",
        "modelVersion": number,
        "modelReleaseDate": "string",
        "modelType": "string",
        "inputFormat": [
            "string"
        ],
        "topicStandard": "string",
        "topicVersion": "string"
    }
]
```

6.3.5 Response Elements

If the action is successful, the service sends back an HTTP 200 response. The API returns either a SynchronousCodeResponse object or an array of SynchronousCodeResponse objects corresponding to the input records.

6.3.6 Errors

The following errors may occur when calling this service:

No models found for topic {topic}
No models are available for the provided topic parameter. Please reach out to your ABS contact to investigate why no model is available.
HTTP Status Code: 500 (*Internal Server Error*)

For information about the errors that are common to all actions, see Section 10, Errors and suggested actions.

6.3.7 Examples

Listing all anzsco models

Sample Request

```
GET /v1/topics/anzsco/models HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json
[
    {
        "modelId": "00000000-0000-0000-0000-000000000002",
        "modelVersion": 2,
        "modelReleaseDate": "2023-12-20T06:06:44.514Z",
        "modelType": "hsvm2",
        "inputFormat": [
            "occp_text",
            "tasks_text"
        ],
        "topicStandard": " ANZSCO - Australian and New Zealand Standard
Classification of Occupations",
        "topicVersion": "2022"
    },
    {
        "modelId": "00000000-0000-0000-0000-000000000001",
        "modelVersion": 1,
        "modelReleaseDate": "2025-05-20T06:06:44.514Z",
        "modelType": "hsvm2",
        "inputFormat": [
            "occp_text",
            "tasks_text"
        ],
        "topicStandard": " ANZSCO - Australian and New Zealand Standard
Classification of Occupations",
        "topicVersion": "2022"
    }
]
```

7. Real-time (Synchronous) Coding

The coding service has been designed to apply a classification code and title to a free text entry. The single record coding feature will enable public facing webforms and other points of data collection to have codes and titles suggested in real time (~1 second).

A small JSON file of up to 300 text records can also be coded synchronously.

Note: when you are running a synchronous small batch, the whole packet needs to be syntactically correct. **If the syntax fails, the whole batch will fail**. As the operation is combined for the whole group of records, none of the records will be able to be coded if there is an error in any record.

When to use synchronous or asynchronous coding

Synchronous coding should only be used for single record coding or small batches of data. If you are coding 900 records, for example, it will be possible to run them in three small batch submissions.

Asynchronous large batch coding is recommended if you need to code or recode a large volume of data. (Large batch coding can be used to code from 1 record to millions of records.)

7.1 Coding against the latest model for a topic

This endpoint is used to code a single or small batch of free text records against the specified coding topic, using the latest model for that topic.

7.1.1 Request Syntax

Depending on whether you are coding a single record or a small batch of records, your request will follow one of the following formats:

1. *Coding a single free text record*

```
POST /v1/topics/{topic}/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: string

{
    "record": {
        "occp_text": "string",
        "tasks_text": "string"
    },
    "numberOfSuggestions": number
}
```

2. *Coding a small batch of free text records*

```
POST /v1/topics/{topic}/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: string
{
    "records": [
        {
            "recordId": "string",
            "occp_text": "string",
            "tasks_text": "string"
        },
    ],
    "numberOfSuggestions": number
}
```

7.1.2 URI Request Parameters

topic

The `uriName` of the topic against which the record is coded. This can be acquired by listing the available topics.
Required: Yes

7.1.3 Request Body

The request accepts the following data in JSON format:

record

The free text record to be coded.

Type: Record object, following the input format specified by the model.
Required: No, but either record or records must be provided.

records

The free text records to be coded.
Type: Array of Record objects, following the input format specified by the model. Each item may optionally specify an additional string value `recordId`.
Length Constraints: Minimum length of 1. Maximum length of 300.
Required: No, but either record or records must be provided.

numberOfSuggestions

The number of suggested codes to be provided if the record cannot be coded successfully. The maximum value of this field is 16.
Type: Number
Required: No

7.1.4 Response Syntax

The response of this endpoint will depend on whether your input request contained a single record or a small batch of records.

1. *Coding a single free text record*

```
HTTP/1.1 200 OK
Content-type: application/json
{
    "codeStatus": "string",
    "input": {
        "occp_text": "string",
        "tasks_text": "string"
    },

    "result": [
        {
            "codeCategory": "string",
            "codeLabel": "string",
            "codeConfidence": number
        }
    ],
}
```

2. *Coding a small batch of free text records*

```
HTTP/1.1 200 OK
Content-type: application/json
[
    {
        "recordId": "string",
        "codeStatus": "string",
        "input": {
            "occp_text": "string",
            "tasks_text": "string"
        },

        "result": [
            {
                "codeCategory": "string",
                "codeLabel": "string",
                "codeConfidence": number
            }
        ],
    }
]
```

7.1.5 Response Elements

If the action is successful, the service sends back an HTTP 200 response. The API returns either a `SynchronousCodeResponse` object or an array of `SynchronousCodeResponse` objects corresponding to the input records.

7.1.6 Errors

The following errors may occur when calling this service:

Malformed record found in request
The free text input did not match the expected format for the model. You can check what the expected format is for a given topic here.
HTTP Status Code: 400 *(Bad Request)*

Batch input contained no records
You tried to code a small batch of records but the records array was empty. Check that you have provided at least one record to be coded and that your request body is correctly formatted.
HTTP Status Code: 400 *(Bad Request)*

Batch records exceeds length limit of 300
You tried to code too many records at once using the synchronous small batch service. Retry with a smaller batch size, or consider using the asynchronous batch coding service.
HTTP Status Code: 400 *(Bad Request)*

There are record(s) outside the min or max char limit
One or more records provided for synchronous coding had too many or too few characters. See 4.2. The error will direct you to the problematic record(s) which should either be excluded or amended to meet the character limits.
HTTP Status Code: 400 *(Bad Request)*

For information about the errors that are common to all actions, see Section 10, Errors and suggested actions.

7.1.7 Examples

Successfully coded a single record using only one free text field

Sample Request

```
POST /v1/topics/osca/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token
{
    "record": {
        "occp_text": "Software developer. Writes code, tests"
    },
    "numberOfSuggestions": 3
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json
```

```
{
    "codeStatus": "successful",
    "input": {
        "occp_text": "Software developer. Writes code, tests"
    },
    "result": [{
        "codeCategory": "261313",
        "codeLabel": "Software Engineer",
        "codeConfidence": 0.6093962788581848
    },{
        "codeCategory": "261312",
        "codeLabel": "Developer Programmer",
        "codeConfidence": 0.43940293565392494
    },{
        "codeCategory": "261399",
        "codeLabel": "Software and Applications Programmers nec",
        "codeConfidence": 0.43756575286388397
    }]
}
```


Successfully coded a single record using all free text fields

Sample Request

```
POST /v1/topics/osca/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token
{
    "record": {
        "occp_text": "software developer",
        "tasks_text": "writing code and unit tests"
    },
    "numberOfSuggestions": 3
}
```

Sample Response


```
HTTP/1.1 200 OK
Content-type: application/json

{
    "codeStatus": "successful",
    "input": {
        "occp_text": "software developer",
        "tasks_text": "writing code and unit tests"
    },
    "result": [{
        "codeCategory": "261313",
        "codeLabel": "Software Engineer",
        "codeConfidence": 0.6093962788581848
    },{
```

```
        "codeCategory": "261312",
        "codeLabel": "Developer Programmer",
        "codeConfidence": 0.43940293565392494
    },{
        "codeCategory": "261399",
        "codeLabel": "Software and Applications Programmers nec",
        "codeConfidence": 0.43756575286388397
    }]
}
```

Unsuccessfully coded a single record using only one free text field

Sample Request

```
POST /v1/topics/osca/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token

{
    "record": {
        "occp_text": "Software developer. Writes code, tests"
    },
    "numberOfSuggestions": 3
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
    "codeStatus": "unsuccessful",
    "input": {
        "occp_text": "Software developer. Writes code, tests"
    },
    "result": []
}
```

Coding a small batch of records

Sample Request

```
POST /v1/topics/osca/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token

{
    "records": [
        {
            "occp_text": "software developer",
```

```
            "tasks_text": "writing code and unit tests"
        }, {
            "occp_text": "Paramedic, respond to emergencies"
        }, {
            "recordId": "1",
            "occp_text": "Sales assistant"
        }
    ],
    "numberOfSuggestions": 3
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json

[
    {
        "codeStatus": "successful",
        "input": {
            "occp_text": "software developer",
            "tasks_text": "writing code and unit tests"
        },
        "result": [{
            "codeCategory": "261313",
            "codeLabel": "Software Engineer",
            "codeConfidence": 0.6093962788581848
        },{
            "codeCategory": "261312",
            "codeLabel": "Developer Programmer",
            "codeConfidence": 0.43940293565392494
        },{
            "codeCategory": "261399",
            "codeLabel": "Software and Applications Programmers",
            "codeConfidence": 0.43756575286388397
        }]
    }, {
        "codeStatus": "unsuccessful",
        "input": {
            "occp_text": "Paramedic, respond to emergencies"
        },
        "result": []
    }, {
        "recordId": "1",
        "codeStatus": "unsuccessful",
        "input": {
            "occp_text": "Sales assistant"
        },
        "result": []
    }
]
```

7.2 Coding against a specific model

This endpoint is used to code a single or small batch of free text records against the specified coding topic, using the specified model.

7.2.1 Request Syntax

Depending on whether you are coding a single record or a small batch of records, your request will follow one of the following formats:

1. Coding a single free text record

```
POST /v1/topics/{topic}/models/{model}/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
   Content-type: application/json
   Authorisation: string
   {
       "record": {
           "occp_text": "string",
           "tasks_text": "string"
       }
       "numberOfSuggestions": number
   }
```

2. Coding records against a specific model

```
POST /v1/topics/{topic}/models/{model}/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
   Content-type: application/json
   Authorisation: string

   {
       "records": [
           {
               "recordId": "string",
               "occp_text": "string",
               "tasks_text": "string"
           }
       ],
       "numberOfSuggestions": number
   }
```

7.2.2 URI Request Parameters

topic

The `uriName` of the topic against which the record is coded. This can be acquired by listing the available topics.
Required: Yes

model

The model GUID for the model you would like to use to code records. This can be acquired by listing the available models for your topic.
Required: Yes

7.2.3 Request Body

The request accepts the following data in JSON format:

record

The free text record to be coded.
Type: Record object, following the input format specified by the model.
Required: No, but either record or records must be provided.

records

The free text records to be coded.
Type: Array of Record objects, following the input format specified by the model. Each item may optionally specify an additional string value `recordId`.
Length Constraints: Minimum length of 1. Maximum length of 300.
Required: No, but either record or records must be provided.

numberOfSuggestions

The number of suggested codes to be provided if the record cannot be coded successfully. The maximum value of this field is 16.
Type: Number
Required: No

7.2.4 Response Syntax

The response of this endpoint will depend on whether your input request contained a single record or a small batch of records.

1. Coding a single free text record

```
HTTP/1.1 200 OK
Content-type: application/json
{
    "codeStatus": "string",
    "input": {
        "occp_text": "string",
        "tasks_text": "string"
    },
    "result": [
        {
            "codeCategory": "string",
            "codeLabel": "string",
            "codeConfidence": number
        }
    ],
}
```

2. Coding a small batch of free text records

```
HTTP/1.1 200 OK
Content-type: application/json
[
    {
        "recordId": "string",
        "codeStatus": "string",
        "input": {
            "occp_text": "string",
            "tasks_text": "string"
        },

        "result": [
            {
                "codeCategory": "string",
                "codeLabel": "string",
                "codeConfidence": number
            }
        ],
    }
]
```

7.2.5 Response Elements

If the action is successful, the service sends back an HTTP 200 response. The API returns either a SynchronousCodeResponse object or an array of SynchronousCodeResponse objects corresponding to the input records.

7.2.6 Errors

The following errors may occur when calling this service:

Malformed record found in request
The free text input did not match the expected format for the model. You can check what the expected format is for a given topic here.
HTTP Status Code: 400 *(Bad Request)*

Batch input contained no records
You tried to code a small batch of records but the records array was empty. Check that you have provided at least one record to be coded and that your request body is correctly formatted.
HTTP Status Code: 400 *(Bad Request)*

Batch records exceeds length limit of 300
You tried to code too many records at once using the synchronous small batch service. Retry with a smaller batch size, or consider using the asynchronous batch coding service.
HTTP Status Code: 400 *(Bad Request)*

There are record(s) outside the min or max char limit

One or more records provided for synchronous coding had too many or too few characters. See Section 4.2. The error will direct you to the problematic record(s) which should either be excluded or amended to meet the character limits.
HTTP Status Code: 400 *(Bad Request)*

For information about the errors that are common to all actions, see Section 10, Errors and suggested actions.

7.2.7 Examples

Successfully coded a single record using all free text fields

Sample Request

```
POST /v1/topics/osca/models/GUID/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token
{
    "record": {
        "occp_text": "software developer",
        "tasks_text": "writing code and unit tests"
    },
    "numberOfSuggestions": 3
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
    "codeStatus": "successful",
    "input": {
        "occp_text": "software developer",
        "tasks_text": "writing code and unit tests"
    },
    "result": [{
        "codeCategory": "261313",
        "codeLabel": "Software Engineer",
        "codeConfidence": 0.6093962788581848
    },{
        "codeCategory": "261312",
        "codeLabel": "Developer Programmer",
        "codeConfidence": 0.43940293565392494
    },{
        "codeCategory": "261399",
        "codeLabel": "Software and Applications Programmers nec",
        "codeConfidence": 0.43756575286388397
    }]
}
```

Coding a small batch of records

Sample Request

```
POST /v1/topics/osca/models/GUID/code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token

{
    "records": [
        {
            "occp_text": "software developer",
            "tasks_text": "writing code and unit tests"
        }, {
            "occp_text": "Paramedic, respond to emergencies"
        }, {
            "recordId": "1",
            "occp_text": "Sales assistant"
        }
    ],
    "numberOfSuggestions": 3
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json

[
    {
        "codeStatus": "successful",
        "input": {
            "occp_text": "software developer",
            "tasks_text": "writing code and unit tests"
        },
        "result": [{
            "codeCategory": "261313",
            "codeLabel": "Software Engineer",
            "codeConfidence": 0.6093962788581848
        },{
            "codeCategory": "261312",
            "codeLabel": "Developer Programmer",
            "codeConfidence": 0.43940293565392494
        },{
            "codeCategory": "261399",
            "codeLabel": "Software and Applications Programmers",
            "codeConfidence": 0.43756575286388397
        }]
    }, {
        "codeStatus": "unsuccessful",
```

```
        "input": {
            "occp_text": "Paramedic, respond to emergencies"
        },
        "result": []
    }, {
        "recordId": "1",
        "codeStatus": "unsuccessful",
        "input": {
            "occp_text": "Sales assistant"
        },
        "result": []
    }
]
```

8. Asynchronous Batch Coding

In addition to real-time coding of single records and small batches of data, the coding service has been designed to code large datasets through asynchronous batching (that is, returning data after a short period of time).

The asynchronous service can be used for as little as one record, up to millions of records.

Note: Asynchronous batch coding should be used if you need to code or recode a large volume of data. While it is the most efficient method of coding larger datasets, it is not real-time, and may be subject to queueing during high load periods.

8.1 Getting an upload URL for input data to a batch coding operation

This endpoint is used to create an asynchronous batch inference operation. The API will return a location where you can upload your input file and begin your batch inference operation.

8.1.1 Request Syntax

Depending on whether you are specifying a model against which to code your records, your request will follow one of the following formats:

1. Coding records against the latest model

```
POST /v1/topics/{topic}/batch-code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: string
```

2. Coding records against a specific model

```
POST /v1/topics/{topic}/models/{model}/batch-code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: string
```

8.1.2 URI Request Parameters

topic

The `uriName` of the topic against which the record is coded. This can be acquired by listing the available topics.
Required: Yes

model

The model GUID for the model you would like to use to code records. This can be acquired by listing the available models for your topic.
Required: No

8.1.3 Request Body

The request does not have a request body.

8.1.4 Response Syntax

```
HTTP/1.1 200 OK
Content-type: application/json

{
    "requestUploadUrl": "string",
    "operationId": "string",
    "bucketKmsKeyArn": "string"
}
```

8.1.5 Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service:

requestUploadUrl
A URL where the records file is to be uploaded.
Type: String

operationId
The identifier of the operation, to be used to check the status of this job. This must be recorded at this point to maintain access to the operation.
Type: String, in GUID format.

bucketKmsKeyArn
A parameter used by the ABS system to ensure the operation's input data is from the same user who created the operation. This must be passed into the x-amz-server-side-encryption-aws-kms-key-id header when uploading your input file.
Type: String

8.1.6 Errors

For information about the errors that are common to all actions, see Section 10, Errors and suggested actions.

8.1.7 Examples

Creating a new operation to code against the latest model for occupation

Sample Request

```
POST /v1/topics/osca/batch-code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-Type: application/json
Authorisation: example token
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
    "requestUploadUrl": "https://domain/endpoint?queries",
    "operationId": "00000000-0000-0000-0000-000000000000",
    "bucketKmsKeyArn": "xyz"
}
```

Creating a new operation to code against a specified model

Sample Request

```
POST /v1/topics/anzsco/models/GUID/batch-code HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-Type: application/json
Authorisation: example token
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
    "requestUploadUrl": "https://domain/endpoint?queries",
    "operationId": "00000000-0000-0000-0000-000000000000",
    "bucketKmsKeyArn": "xyz"
}
```

8.2 Uploading data for inference

Once you have created an inference operation, you will need to upload your data to the provided requestUploadUrl. This is a pre-signed HTTP request which is managed by the AWS S3 server, and the expected input is outlined below.

8.2.1 Request Syntax

```
PUT requestUploadUrl HTTP/1.1
x-amz-server-side-encryption: aws:kms
x-amz-server-side-encryption-aws-kms-key-id: string

{ "recordId": "string", "occp_text": "string", "tasks_text": "string" }
...
```

8.2.2 URI Request Parameters

requestUploadUrl
The location where the input file is being uploaded. This is provided when you first create the inference operation.
Type: String

8.2.3 Request Header Parameters

Please note: the `x-amz-server-side-encryption` header is not variable and should always have the value `aws:kms`.

x-amz-server-side-encryption-aws-kms-key-id
A parameter used by the ABS system to ensure the input data is from the same user who created the operation. This is provided in the bucketKmsKeyArn field when you first create your inference operation.
Type: String

8.2.4 Request Body

The request accepts your input file in JSONL format. The maximum input file size is 5GB. All lines of input must contain the same fields, and these fields should satisfy the Record type for the relevant topic/model as specified when creating the upload URL. You may specify the additional field outlined below.

recordId
An identifier for the record being coded. This need not be unique.
Type: String
Required: No

8.2.5 Response Syntax

```
HTTP/1.1 200 OK
```

8.2.6 Errors

For information about the errors that are common to all actions, see Section 10, Errors and suggested actions.

8.2.7 Examples

Specifying all free text inputs and a record identifier

Sample Request

```
PUT https://domain/endpoint?queries HTTP/1.1
x-amz-server-side-encryption: aws:kms
x-amz-server-side-encryption-aws-kms-key-id: xyz

{ "recordId": "1", "occp_text": "software developer", "tasks_text": "writing
code and unit tests" }
```

```
{ "recordId": "2", "occp_text": "Paramedic", "tasks_text": "responding to
medical emergencies" }
...
```

Sample Response

```
HTTP/1.1 200 OK
```

Specifying a single free text input and a record identifier

Sample Request

```
PUT https://domain/endpoint?queries HTTP/1.1
x-amz-server-side-encryption: aws:kms
x-amz-server-side-encryption-aws-kms-key-id: xyz

{ "recordId": "1", "occp_text": "software developer, writes code and unit
tests" }
{ "recordId": "2", "occp_text": "Paramedic, respond to emergencies" }
...
```

Sample Response

```
HTTP/1.1 200 OK
```

Specifying all free text inputs and no record identifier

Sample Request

```
PUT https://domain/endpoint?queries HTTP/1.1
x-amz-server-side-encryption: aws:kms
x-amz-server-side-encryption-aws-kms-key-id: xyz

{ "occp_text": "software developer", "tasks_text": "writing code and unit
tests" }
{ "occp_text": "Paramedic", "tasks_text": "responding to medical emergencies" }
...
```

Sample Response

```
HTTP/1.1 200 OK
```

8.3 Checking the status of a batch inference operation

This endpoint is used to check the status of your batch inference job. When the status of your job is complete, the service will return a URL to copy into your web browser to retrieve your coded data.

8.3.1 Request Syntax

Depending on whether you are specifying a model against which to code your records, your request will follow one of the following formats. The application backend handles these requests

identically, so you don't need to worry about recording the model which you used when you began the operation.

1. Checking an operation by specifying the topic only

```
GET /v1/topics/{topic}/batch-code/operations/{operation_id} HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: string
```

2. Checking an operation by specifying both the topic and model

```
GET /v1/topics/{topic}/models/{model}/batch-code/operations/{operation_id}
HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: string
```

8.3.2 URI Request Parameters

`topic`
The `uriName` of the topic against which the record is coded. This can be acquired by listing the available topics.
Required: Yes

`model`
The model GUID for the model you would like to use to code records. This can be acquired by listing the available models for your topic.
Required: No

`operation_id`
The GUID of the operation to get the status of. This value is provided when you first create your inference operation.
Required: Yes

8.3.3 Request Body

The request does not have a request body.

8.3.4 Response Syntax

```
HTTP/1.1 200 OK
Content-type: application/json

{
   "operationStatus": "string",
   "responseDownloadUrl": "string",
   "error": "string"
}
```

8.3.5 Response Elements

If the specified operation exists, the service sends back an HTTP 200 OK status code. The status of the operation will dictate the contents of the response. This data is returned in JSON format by the service:

`operationStatus`
The status of the operation.
Type: String
Valid Values: `awaiting_input` | `in_progress` | `complete` | `timed_out` | `failed`

`responseDownloadUrl`
A URL where the output data file can be downloaded. This field is optional and is returned only if `operationStatus` is `complete`.
Type: String

`metadataDownloadUrl`
A URL where the output metadata file can be downloaded. This file includes information about the model used to code your data.This field is optional and is returned only if `operationStatus` is `complete`.
Type: String

`error`
Information on why the operation failed. This field is optional and is returned only if `operationStatus` is `failed`.

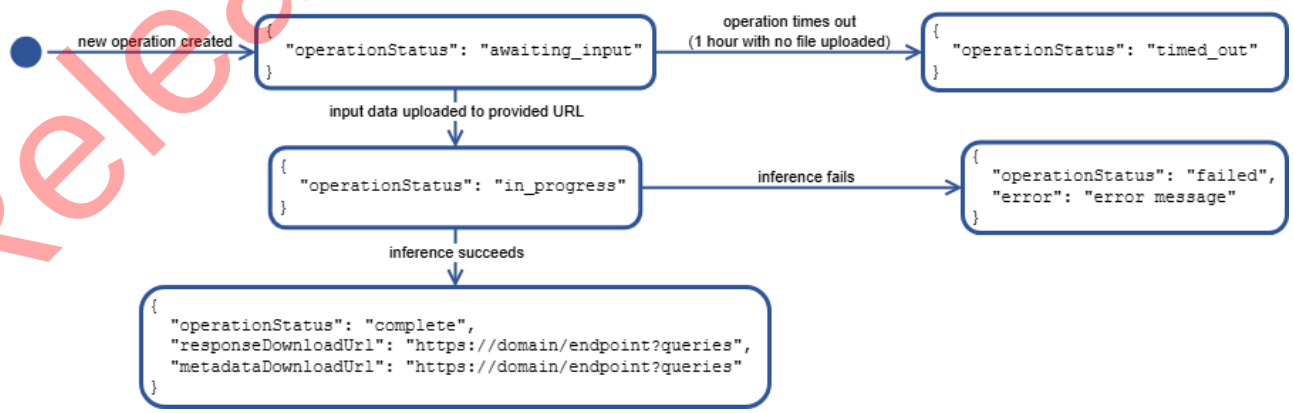
A note about presigned URLs

The `responseDownloadUrl` and `metadataDownloadUrl` are presigned URLs.

Anyone with this link will be able to download your output file, but the link will expire after one hour, after which you will have to get a new URL for your output file.

Your output files will be deleted from the system within 24 hours after your inference operation succeeds.

A state machine indicating the progression of operations is shown below:



8.3.6 Errors

The following errors may occur when calling this service:

Unable to retrieve operation for given id

No operations were found to match the given operation_id. Please confirm your operation ID. If you have lost your operation ID, you will have to create a new operation.
HTTP Status Code: 404 *(Not Found)*

User is not authorised to retrieve operation GUID

The specified operation does not belong to the current user.
You may have authenticated with the wrong user or specified the wrong operation_id. Try authenticating again with the right credentials, and confirm your operation.
HTTP Status Code: 401 *(Unauthorised)*

For information about the errors that are common to all actions, see Section 10, Errors and suggested actions.

8.3.7 Examples

Getting the status of an operation

Sample Request

```
GET /v1/topics/osca/batch-code/operations/GUID HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token
```

Sample Request, specifying the model used

```
GET /v1/topics/anzsco/models/GUID/batch-code/operations/GUID HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Content-type: application/json
Authorisation: example token
```

Sample Responses

```
HTTP/1.1 200 OK
Content-type: application/json
```

Request sample	***Expected response body***	***Interpretation***
New operation (data not yet uploaded)	{ "operationStatus": "awaiting_input" }	• The server acknowledges the operation exists. • No input data file has yet been received. • The operation is pending your next action - typically an upload via PUT request.

Just uploaded	<pre>{
	"operationStatus":
	"in_progress"
	}</pre>
		• The server has received the input data file.
		• The specified operation is now running, or may be queued to run soon.
		• You should keep checking in periodically (for example, up to once every ten minutes) to see how the operation progresses.
		• The output files will be deleted within 24 hours.

Never uploaded	<pre>{
	"operationStatus":
	"timed_out"
	}</pre>
		• The server acknowledges the operation exists.
		• No input data has yet been received.
		• The operation has timed out due to inactivity and can no longer accept input data.
		• If you wish to run an asynchronous batch operation, you will need to create a new operation.

Operation complete	<pre>{
	"operationStatus":
	"complete",
	"responseDownloadUrl":
	"https://domain/endpoint?queries",
	"metadataDownloadUrl":
	"https://domain/endpoint?queries",
	}</pre>
		• The specified operation is now complete.
		• The output files are now available at the provided URLs.
		• You should download the output files now as they will be deleted within 24 hours.
		• There may be unsuccessfully coded records in the output file. Errors will be reported on a record-by-record basis where possible. This reduces the need to recode the entire input file.

Operation failed	<pre>{
	"operationStatus":
	"failed",
	"error": "error message"
	}</pre>
		• The specified operation has failed inference.
		• Check your input file for any errors or invalid records and try again.
		• The error message may provide context on what caused the operation failure. If the error message does not help resolve the issue, please note your operation id when contacting us for support.
		• If you wish to run another asynchronous batch operation, you will need to create a new operation.

8.4 Downloading processed data from a complete operation

Once your asynchronous inference operation is complete, you can download the output file by accessing (copying into a web browser) the responseDownloadUrl that is provided when you check the status of a complete operation. The same process may be used to view the operation metadata, available at the metadataDownloadUrl.

This is a generic HTTP GET request which is managed by the AWS S3 server, and the expected format is outlined below.

8.4.1 Response Elements

The asynchronous batch coding service outputs a jsonl file with each line corresponding to the record from the original input file. Each line is an AsynchronousCodeResponse object.

8.4.2 Examples

In response to input which specifies a record identifier

Sample Request

```
GET https://domain/endpoint?queries HTTP/1.1
```

Sample Response

```
HTTP/1.1 200 OK
Date: Thu, 20 Jun 2024 02:26:34 GMT
Last-Modified: Thu, 20 Jun 2024 02:24:04 GMT
Accept-Ranges: bytes
Content-Type: application/octet-stream
Server: AmazonS3
Content-Length: 7660
...

{ "recordId": "1", "result": { "codeCategory": "261313", " codeLabel":
"Software Engineer", "codeConfidence": 0.98 } }
{ "id": "2", "suggestions": [{ "codeCategory": "411111", "codeLabel":
"Ambulance Officer", "codeConfidence":  0.26 }, { "codeCategory": "411112",
"codeLabel": "Intensive Care Ambulance Paramedic", "codeConfidence": 0.24 }] }
...
```

In response to input which specifies no record identifier

Sample Request

```
GET https://domain/endpoint?queries HTTP/1.1
```

Sample Response

```
HTTP/1.1 200 OK
Date: Thu, 20 Jun 2024 02:26:34 GMT
Last-Modified: Thu, 20 Jun 2024 02:24:04 GMT
```

```
Accept-Ranges: bytes
Content-Type: application/octet-stream
Server: AmazonS3
Content-Length: 7660
...

{ "recordId": "", "result": { "codeCategory": "261313", " codeLabel": "Software
Engineer", "codeConfidence": 0.98 } }
{ "recordId": "", "suggestions": [{ "codeCategory": "411111", "codeLabel":
"Ambulance Officer", "codeConfidence":  0.26 }, { "codeCategory": "411112",
"codeLabel": "Intensive Care Ambulance Paramedic", "codeConfidence": 0.24 }] }
...
```

9. Reporting Issues

If you encounter bugs or have feedback on the service, please report these via the following mechanism:

9.1 Request Syntax

```
GET /v1/security.txt HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Authorisation: string
```

9.2 URI Request Parameters

The request does not use any URI parameters.

9.3 Request Body

The request does not have a request body.

9.4 Response Syntax

```
HTTP/1.1 200 OK
Content-type: text/html

<information on reporting errors>
```

9.5 Example

Sample Request

```
GET /v1/security.txt HTTP/1.1
Host: https://partner-coder.api.abs.gov.au
Authorisation: example token
```

Sample Response

```
HTTP/1.1 200 OK
Content-type: text/html
<p>contact: mailto:security@abs.gov.au
expires: 2024-06-27T05:45:00.000Z</p>
```

10. Errors and suggested actions

These codes help identify issues on both the client and server sides, allowing for troubleshooting and resolution of HTTP request problems.

Error Message	Why this happened	You should…
Generic errors possible on all api calls		
Invalid request body HTTP Status Code: **400**	Something was wrong with your request body syntax. Example: small batch records exceed length limit of 300/you tried to code too many records at once using the synchronous small batch service.	• check the request body syntax for the API call and try again. • Remove any special characters from free text inputs (see section 4.2) • If your small batch has this error, retry with a smaller batch size, or consider using the asynchronous batch coding service.
User is not authorized to access this resource with an explicit deny HTTP Status Code: **403**	You have either not authenticated or your authentication token has expired.	• authenticate again.
Error performing request HTTP Status Code: **500**	This happens when something unexpected goes wrong on the server. In some instances, further context is provided.	• retry after a short delay; report persistent issues to support.
Problems selecting model/topic		
The specified topic does not exist HTTP Status Code: **404**	No topics matched the provided topic parameter.	• check the available topics before proceeding.
The specified model does not exist HTTP Status Code: **404**	No models matched the provided model parameter. You'll see this if the system can't locate the specified model - maybe due to a typo or outdated ID.	• verify that the model name or ID is correct and still active. • check which models are available or use the default (latest) model for the topic.
Selected model does not match the input topic. HTTP Status Code: **409**	The given model GUID does not correspond to the specified coding topic.	• check which models are available or use the default (latest) model for the topic.
Errors on the get models endpoint		

No models found for topic *topic* HTTP Status Code: **500**	No models are available for the provided topic parameter.	• reach out to your ABS contact to investigate why no model is available.
Synchronous coding errors		
Malformed record found in request HTTP Status Code: **400**	The free text input did not match the expected format for the model.	• check what the expected format is for a given topic here.
Batch input contained no records HTTP Status Code: **400**	You tried to code a small batch of records but the records array was empty.	• check that you have provided at least one record to be coded, and • check that your request body is correctly formatted.
There are record(s) outside the min or max char limit: Record with index *x* has a total text length under 3 min Record with index *y* has a total text length over 100 max … HTTP Status Code: **400**	One or more records provided had too many or too few characters. See 4.2.	• check that each record to be coded has 3-100 (inclusive) characters across the two input fields.
RateLimitExceededException HTTP Status Code: **429** *(Too Many Requests)*	You'll see this when sending too many requests too quickly.	• space out your requests and implement retry logic.i
Asynchronous coding errors		
User is not authorised to retrieve operation *GUID* HTTP Status Code: **401**	The specified operation does not belong to the current user. You may have authenticated with the wrong user or specified the wrong operation_id.	• authenticate again and check you have the right credentials, and • confirm your operation id.
Unable to retrieve operation for given id HTTP Status Code: **404**	No operations were found to match the given operation_id. This is most likely due to a typo.	• confirm the operation ID is correct. • if you have lost your operation ID, you will have to create a new operation.

See more information on HTTP errors at HTTP response status codes - HTTP | MDN.

11. Glossary of Inputs and Responses

11.1 Model details

These fields are returned by various methods in Section 6.

Field	Description	Type
modelId	Unique identifier used to reference the ML model.	String (GUID format)
modelVersion	Version number of the model trained on the topic. Distinct from topicVersion.	Number
modelReleaseDate	Date the model was released, in ISO8601 format.	String
modelType	ML algorithm used (e.g., hsvm).	String
inputFormat	List of expected input field names.	Array of strings
topicStandard	Full name of the classification topic.	String
topicVersion	Version number of the topic classification used in model training.	String

11.2 RecordObject

These are the fields expected when submitting data to the coding service.

Field	Description	Type
occp_text	Free-text description of an occupation.	String
tasks_text	Tasks or duties related to the occupation.	String
recordId	Optional identifier for each input record.	String

11.3 Response Objects

These are returned after synchronous or asynchronous coding operations.

11.3.1 SynchronousCodeResponse

Field	Description	Type
recordId	Identifier for the submitted input record (if originally provided).	String
codeStatus	Coding outcome. Valid values: successful, unsuccessful.	String
input	Input record submitted to the model.	RecordObject
result	List of predicted codes and labels. *Min length: 0; Max: 16 (or value of numberOfSuggestions)*	Array of CodedRecord

11.3.2 AsynchronousCodeResponse

Field	Description	Type
recordId	Identifier for the record or empty string if none provided.	String

result	Top code assigned if coding was successful.	CodedRecord object
suggestions	List of alternate codes if coding was unsuccessful. *Min length: 1; Max: 3*	Array of CodedRecord

11.3.3 CodedRecord

Field	**Description**	**Type**
codeCategory	Code assigned to the input.	String
codeLabel	Description of the code category.	String
codeConfidence	Confidence score (e.g., 0.92). May be rounded or multiplied by 100 for a percentage.	

12. Security

The WoAG Coding Service and API has been security assessed by an independent registered assessor within the Australian Signals Directorate (ASD) Information Security Registered Assessors Program (IRAP) Program. This assessment found the WoAG Coding Service and API to have met the control and security objectives defined through the Australian Government Information Security Manual (ISM).

Agencies may need to sign off in-house on using an external API, for business, legal, or security reasons. They may also need to check on their own behalf that the API response is from the address they sent the request to.

The following security controls, drawn from the ISM, are included to assist partner agencies in assessing their risks when using this service.

Control Name	System Security Controls
Cryptography	• Data is encrypted in transit to and from the API. All APIs created with Amazon API Gateway expose HTTPS endpoints only. API Gateway does not support unencrypted (HTTP) endpoints. • API Gateway has been configured to choose a minimum Transport Layer Security (TLS) protocol version of TLS 1.2. • Data is encrypted at rest if it is to be stored by the request action; only file-based batch requests require the storage of request data. KMS Keys will be created for each environment and applied to data (or metadata) storage components e.g., S3 buckets, DynamoDB tables.
Data Transfers	• Bulk data transfers only occur for asynchronous requests. This occurs through the AWS WAF and the AWS ABS Gateway. • Data transferred as part of an asynchronous request is scanned. • Resources involved in the coding of file-based requests (e.g. Lambda, SageMaker) are able to read data from bucket(s) containing post-scanned/validated data, not raw ingress data, and are only able to write to a specific egress bucket. • For file-based batch coding, consumers are provided S3 pre-signed URLs for data file ingress and egress. This aligns with ISM cryptographic control requirements (ISMF 1123–1126), access control measures for external interfaces (ISMF 1295–1300), and secure transmission/storage of sensitive data (ISMF 1352–1354). • These URLs are configured with expiration times aligned with the time required to perform a coding request. This is a dynamic value and is configured based on performance data from implemented models. • Data stored in service of file-based batch coding (e.g. ingressed request data and egressed coded response data) is configured for automated expiry (deletion). Minimal expiry time is configured based on the time required to perform a coding request, pending performance data from implemented models.
Data Sovereignty	• No data will be stored or processed outside Australia. • Services will never failover to services outside of Australia.

Control Name	System Security Controls
Machine Learning (ML)	• There is no external connection (outside of the dedicated ABS accounts) or other ML reference used in the WoAG Coding Service or in the training of the ML models. • Only isolated instances of Machine Learning within ABS-owned secure AWS accounts are used to train the models that underpin the Coding Service. • While the ABS is exploring the use of Distilbert - Large Language Models (LLM) combined with Census data to train coder models, only more traditional ML models such as Hierarchical Support Vector Machine (HSVM) models, trained only using Census data, will be used in the external service. • Only specific response text, separated from all other response data, is used to create the ML models and in the Coding Service. • The service can only respond with the classification codes and labels defined in the relevant classification standard and version, unless a record identifier is also provided by the user. In this instance, the record identifier is returned to the user with the data. • The application of the models used in the Coding Service, and all data passed through the coders, remains within the ABS secure accounts at all times. No user data is stored or retained. User data is temporarily stored within ABS secure accounts while being processed, and then deleted.