Boldon James
A QINETIQ company

Classifier API for Mac

UM643700

November 18

Gold Microsoft Partner

Customer Documentation

Contents

1 INTRODUCTION

Boldon James Classifier API enables limited Classifier functionality to be utilized programmatically. The API functionality includes reading, editing, comparing and applying labels to files as well as querying available selectors and values in a Classifier configuration.

Note: ***SISL*** refers to the internal encoded format of a Classifier classification (label).

1.1 Classifier Policy Control

The Classifier API operates using the current Classifier label configuration. Refer to the Classifier Administration Guide for further information.

2 INSTALLATION

Run the supplied installation file (BJClassifierAPI.pkg), which will install the Classifier API into the /Library/Frameworks directory.

3 API METHODS

3.1 Object Instantiation

The Classifier API is packaged as a framework which exposes functionality as both C and Objective-C APIs. In order to use it from a project in XCode:

- Add a reference to BJClassifierAPI.framework to the project
- Add an @import BJClassifierAPI or #include <BJClassifierAPI/BJClassifierAPI.h> to the calling source file
- Create an instance of the API protocol using one of the factory functions.

The **Create** functions are responsible for loading the Classifier configuration. The calls can fail if a valid Classifier configuration cannot be found or if the licence contents are invalid.

Note: The API looks for the configuration in the same way as the Classifier Client or, if usingServiceMode is set to true, like a service product. Refer to the Classifier Administration Guide for more information.

Using Objective-C:

```
+(id<BJClassifierAPIProtocol>) createAPIWithCulture:(NSString*)cultureName andLicenceText:(NSString*) licenceText error:(NSError**)error;

+(id<BJClassifierAPIProtocol>) createAPIWithCulture:(NSString*)cultureName andLicenceText:(NSString*)licenceText usingServiceMode:(BOOL)serviceMode error:(NSError**)error;
```

Using either C or Objective-C:

```
BJClassifierAPIRef BJClassifierAPICreate(CFStringRef cultureName, CFStringRef licenceText, CFErrorRef* error);

BJClassifierAPIRef BJClassifierAPICreateWithServiceMode(CFStringRef cultureName, CFStringRef licenceText, bool usingServiceMode, CFErrorRef* error);
```

Parameters:

[in] cultureName

Type: string

Description: Name of the culture to use (e.g. “EN-US”, “FR”). If this parameter is null or empty, the API will use the current thread culture. The parameter is only required with multiple-language Classifier configurations.

[in] licenceString

Type: string

Description: A string of the contents from a valid Classifier licence file (impCPI.lic).

[in] usingServiceMode

Type: BOOL

Description: Specify if you want the API to run in Service Mode. Default is NO.

[out, optional] error

Type: error**

Description: If an error occurs, gets populated with extended error information

Return Value:

An instance of the C or Objective-C api type on success.

If the call failed, returns nil. If specified, error may be one of

- **BadCulture**.
- **Unlicenced**.
- **InvalidConfig**.

For example, using Objective-C

```
NSString* licenceText = [NSString stringWithContentsOfFile:@"impCPI.lic"
encoding:NSASCIIStringEncoding error:nil];

id apiRef = [BJClassifierAPI createAPIWithCulture:@"" andLicenceText:licenceText error:nil];
```

or using C

```
CFStringRef licenceText = getLicenceString();

BJClassifierAPIRef apiRef = BJClassifierAPICreate(CFSTR(""),licenceText, NULL);
```

3.2 Threading

Prior to calling any API functions on a thread other than the main one, the thread must be attached as follows:

Using Objective-C, using the BJClassifierAPI class:

```
+(void*) attachCurrentThread;
```

Using either C or Objective-C:

```
void* BJClassifierAttachCurrentThread();
```

Return Value:

Type: void*

A reference to the attached thread.

If you no longer need to use an attached thread, that thread may be detached, as follows:

Using Objective-C, using the BJClassifierAPI class:

```
+(void) detachThread:(void*)threadHandle;
```

Using either C or Objective-C:

```
void BJClassifierDetachThread(void* threadHandle);
```

Parameters:

[in] threadHandle

Type: void*

Description: the thread to detach, as returned from the attach functions.

3.3 Getting the classification of a file

The **CopyClassificationFromFile** functions are used to obtain the Classifier classification (as a string) from the given file.

Using Objective-C:

```
-(NSString*) copyClassificationFromFileAtPath:(NSString*)path error:(NSError**)error;

-(NSString*) copyClassificationFromFileWithHandle:(int)handle andPath:(NSString*)path error:(NSError**)error;
```

Using either C or Objective-C:

```
CFStringRef BJClassifierCopyClassificationFromFileAtPath(BJClassifierAPIRef classifierAPI, CFStringRef path,
CFErrorRef* error);

CFStringRef BJClassifierCopyClassificationFromFileWithHandle(BJClassifierAPIRef classifierAPI, int handle,
CFStringRef path, CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] handle

Type: int

Description: File handle that will be used to obtain the classification, as returned from open(). If the file type is not supported, fileName is used to get the marking from extended file attributes.

[in] path

Type: string

Description: Path of the file from which to obtain a Classifier classification.

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: string

On success, Classifier classification in its SISL format. On Error, nil.

If the call failed, the following may be specified in error:

- **Uninitialised**
- **Unlicenced**
- **InvalidFile**
- **InvalidHandle**
- **CannotDecodeLabel**
- **UnsupportedFileType**

3.4 Getting the names of the selectors in a label

The **copyLabelSelectorNames** functions are used to obtain a string array of all selector names in a given classification.

Using Objective-C:

```
-(NSArray<NSString*>*) copyLabelSelectorNamesFromSISL:(NSString*)sisl error:(NSError**)error
```

Using either C or Objective-C:

```
CFArrayRef BJClassifierCopyLabelSelectorNamesFromSISL(BJClassifierAPIRef classifierAPI, CFStringRef sisl,
CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] sisl

Type: string

Description: Classifier classification in its own internal label format. This string can be obtained from a call to copyClassification

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: an array of strings

On success, a string array of the names of all selectors within the SISL. On error, nil.

If the call failed, the following may be returned in error:

- **Uninitialised**
- **Unlicenced**
- **CannotDecodeLabel.**

3.5 Getting the values from a label

The **copyLabelValues** functions are used to obtain a string array of all selector values in a given classification.

Using Objective-C:

```
-(NSArray<NSString*>*) copyLabelValuesFromSISL:(NSString*)sisl error:(NSError**)error;
```

Using either C or Objective-C:

```
CFArrayRef BJClassifierCopyLabelValuesFromSISL(BJClassifierAPIRef classifierAPI, CFStringRef sisl, CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] sisl

Type: string

Description: Classifier classification in its own internal label format. This string can be obtained from a call to copyClassification

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: an array of strings

On success, a string array of all values within the SISL. On error, nil.

If the call failed, the following may be returned in error

- **Uninitialised**
- **Unlicenced**
- **CannotDecodeLabel**

3.6 Getting the value for a single select selector

The **copyValueForSelector** functions are used to obtain the value for a given selector from a label in SISL format.

Using Objective-C:

```
-(NSString*) copyValueForSelectorWithName:(NSString*)selectorName fromSISL:(NSString*)sisl error:(NSError**)error;
```

Using either C or Objective-C:

```
CFStringRef BJClassifierCopyValueForSelectorWithName(BJClassifierAPIRef classifierAPI, CFStringRef sisl,
CFStringRef selectorName, CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] selectorName

Type: string

Description: Name of a selector in the Classifier configuration. Alternate names not applicable.

[in] sisl

Type: string

Description: Label classification in SISL format. This string can be obtained from a call to **copyClassification**

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: string

On success, the language-specific name of the value in the specified Classifier classification for the specified selector. On error, nil.

If the call failed, the following may be returned in error:

- **Uninitialised**
- **Unlicenced**
- **InvalidSelector**

- **WrongSelectorType**
- **CannotDecodeLabel**

3.7 Getting the value for a multi select selector

The **copyValueForMultiSelector** functions are used to obtain the value for a given multi-selector from a Classifier classification.

Using Objective-C:

```
-(NSArray<NSString*>*) copyValuesForMultiSelectorWithName:(NSString*)selectorName fromSISL:(NSString*)sisl
error:(NSError**)error;
```

Using either C or Objective-C:

```
CFArrayRef BJClassifierCopyValueForMultiSelectorWithName(BJClassifierAPIRef classifierAPI, CFStringRef sisl,
CFStringRef selectorName, CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] selectorName

Type: string

Description: Name of a selector in the Classifier configuration. Alternate names not applicable.

[in] SISL

Type: string

Description: Classifier classification in its SISL label format. This string can be obtained from a call to **copyClassification**

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: array of strings

On success, the language-specific, if applicable, name(s) of the value in the specified Classifier classification for the specified selector. On error, nil.

If the call failed, the following may be returned in error:

- **Uninitialised**
- **Unlicenced**
- **SelectorNotFound**
- **InvalidSelector**
- **WrongSelectorType**

- **CannotDecodeLabel**

3.8 Setting the classification of a file

The **SetClassificationOnFile** functions are used to set the Classifier classification for the given file.

Using Objective-C:

```
-(BOOL)setClassificationOnFileAtPath:(NSString*)path toSISL:(NSString*)sisl
withRefreshRequired:(BOOL)refreshRequired error:(NSError**)error;

-(BOOL)setClassificationOnFileWithHandle:(int)handle andPath:(NSString*)path toSISL:(NSString* )sisl
withRefreshRequired:(BOOL)refreshRequired error:(NSError**)error;
```

Using either C or Objective-C:

```
bool BJClassifierSetClassificationOnFileAtPath(BJClassifierAPIRef classifierAPI, CFStringRef fileName, CFStringRef
SISL, bool refreshRequired, CFErrorRef*error);

bool BJClassifierSetClassificationOnFileWithHandle(BJClassifierAPIRef classifierAPI, int handle, CFStringRef fileName,
CFStringRef SISL, bool refreshRequired, CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] handle

Type: int

Description: File handle that will be used to set the classification, as returned from open(). If the file is not supported, path will be used instead.

[in] path

Type: string

Description: Path of the file for which the Classifier classification is to be set.

[in] sisl

Type: string

Description: The classification in SISL format. This string can be derived using other methods on the API.

[in] refreshRequired

Type: BOOL

Description: Sets a property on the file. If **YES**, Office Classifier will update any visual markings (e.g. header/footer) the next time that the document is opened. This ensures that the visual markings on the document match the updated classification.

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: BOOL

Yes if the call succeeded, **NO** if not

If NO is returned, the following may be returned in error:

- **Uninitialised**
- **Unlicenced**
- **InvalidFile**
- **InvalidHandle**
- **UnsupportedFileType**
- **CannotDecodeLabel**
- **CantWriteToFile**
- **CantWriteViaHandle**

3.9 Setting the value of a single select selector

These functions are used to set the value for a given selector in a Classifier classification. In order to clear the value for a selector, a null/empty string should be specified for the value.

Using Objective-C:

```
-(BOOL) newSISLWithSISL:(NSString*)currentSISL andSelector:(NSString*)selectorName
withValue:(NSString*)selectorValue newSISL:(NSString**) newSISL error:(NSError**)error;
```

Using either C or Objective-C:

```
BJClassifierCreateSISLWithSISLAndSelector(BJClassifierAPIRef classifierAPI, CFStringRef currentSISL, CFStringRef
selectorName, CFStringRef selectorValue, CFStringRef* newSISL, CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] currentSISL

Type: string

Description: Label classification in SISL format. This string can be obtained from a call to **CopyClassification**.

[in] selectorName

Type: string

Description: Name of a selector in the Classifier configuration.

[in] selectorValue

Type: string

Description: The name of the value in the specified Classifier classification for the specified selector that should be set in the Classifier classification.

This parameter must match the name value in the Culture the API is operating in.

Note: For 'Date Picker' selectors, the value should be in the form "yyyyMMdd"

[out] newSISL

Type: string

Description: The updated Classifier classification in its own internal label format after setting the specified selector value.

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: bool

true if the call succeeds, false if an error occurs.

If the call failed, the following may be returned in error:

- **Uninitialised**
- **Unlicenced**
- **InvalidSelector**
- **InvalidSelectorValue**
- **CannotDecodeLabel**
- **WrongSelectorType**
- **SelectorNotFound**

3.10 Setting the value of a multi-select selector

These functions are used to set the values for a given multi-select selector in a Classifier classification. In order to clear the value for a selector, a null/empty string should be specified as the value.

Using Objective-C:

```
-(BOOL) newSISLWithSISL:(NSString*)currentSISL andMultiSelector:(NSString*)selectorName
withValue:(NSArray<NSString*>*)selectorValues newSISL:(NSString**) newSISL error:(out NSError**)error;
```

Using either C or Objective-C:

```
bool BJClassifierCreateSISLWithSISLAndMultiSelector(BJClassifierAPIRef classifierAPI, CFStringRef currentSISL,
CFStringRef selectorName, CFArrayRef selectorValues, CFStringRef* newSISL, CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] currentSISL

Type: string

Description: Classifier classification in SISL format. This string can be obtained from a call to **CopyClassification**

[in] selectorName

Type: string

Description: Name of a selector in the Classifier configuration.

[in] selectorValues

Type: string[]

Description: Name(s) of the values in the specified classification that should be set for the specified selector.

The selector value names should match their names in the culture the API was initialised with.

[out] newSISL

Type: string

Description: The updated Classifier classification in SISL format after setting the specified selector value.

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: bool

true if the call succeeds, false if an error occurs.

If the call failed, the following may be returned in error:

- **Uninitialised**
- **Unlicenced**
- **InvalidSelector**
- **InvalidSelectorValue**
- **WrongSelectorType**
- **SelectorNotFound**
- **CannotDecodeLabel**

3.11 Getting the available selectors

The **CopySelectorObjects** functions will return an array of all selector objects in the label configuration.

Using Objective-C:

```
-(NSArray<id<BJClassifierSelectorProtocol>>*) copySelectorObjects:(NSError**)error;
```

Using either C or Objective-C:

```
CFArrayRef BJClassifierCopySelectorObjects(BJClassifierAPIRef classifierAPI, CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

An array of id<BJClassifierSelectorProtocol> (Objective-C) or BJClassifierSelectorRef (C) if the call succeeds, or NULL if an error occurs.

If the call failed, the following may be returned in error:

- **Uninitialised**
- **Unlicenced**
- **SelectorNotFound**

3.12 Getting the available values for a selector

The **CopyValueObjects** functions will return an array of all selector values available for a specific selector.

The object's values will match the culture that the API is using.

Using Objective-C:

```
-(NSArray<id<BJClassifierSelectorValueProtocol>>*) copyValueObjectsforSelectorWithID:(NSString*)selectorID
error:(NSError**)error
```

Using either C or Objective-C:

```
CFArrayRef BJClassifierCopyValueObjectsForSelectorWithID(BJClassifierAPIRef classifierAPI, CFStringRef selectorID,
CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] SelectorID

Type: string

Description: ID of the selector to return the values of. The selector ID can be determined by retrieving the identifier property of a BJClassifierSelectorRef or id<BJClassifierSelectorProtocol> object.

Return Value:

id<BJClassifierSelectorValueProtocol> (Objective-C) or BJClassifierSelectorValueRef (C) associated with a specific selector if the call succeeds, or NULL if an error occurs.

If the call failed, the following may be returned in error:

- **Uninitialised**
- **Unlicenced**
- **InvalidSelector**
- **SelectorNotFound**

3.13 Comparing labels

Compare two labels in SISL format to determine if they match, if one label is greater or if they are not comparable.

Using Objective-C:

```
- (int32_t)compareSISL:(NSString *)sisl1 withSISL:( NSString *)sisl2;
```

Using either C or Objective-C:

```
int32_t BJClassifierCompareLabels(BJClassifierAPIRef classifierAPI, CFStringRef sisl1, CFStringRef sisl2);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] sisl1

Type: string

Description: first SISL for comparison.

[in] sisl2

Type: string

Description: second SISL for comparison.

Return Value:

Type: int, may be one of the following

LabelsAreEqual if the labels are the same.

FirstLabelDominates if sisl1 is greater than sisl2.

SecondLabelDominates if sisl1 is less than sisl2.

LabelsAreDifferent if the labels are not the same and neither is greater.

If the call failed, the following may be returned:

- **CannotDecodeLabel**
- **Unlicenced**
- **Uninitialised**

3.14 Calculating a High-water mark

The high-water mark functions take a collection of labels in SISL format and output a SISL representing the high-water mark.

The high-water mark label contains the highest present selector value for each selector in use in a collection of labels.

Using Objective-C:

```
-(NSString*) newHighWaterMarkWithLabels:(NSArray<NSString*>*)labels error:(NSError**)error;
```

Using either C or Objective-C:

```
CFStringRef BJClassifierCreateHighWatermarkFromLabels(BJClassifierAPIRef classifierAPI, CFArrayRef labels,
CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[in] labels

Type: an array of strings

Description: Array of strings in SISL format that are used to obtain the high-water mark.

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: string

If the function succeeds, a SISL string that represents the high-water mark of the array of labels. If an error occurs, NULL.

If the call failed, the following may be returned in error:

- **CannotCalculateHighWatermark**
- **InvalidCollection**
- **CannotDecodeLabel**
- **Unlicenced**
- **Uninitialised**

3.15 Refreshing the configuration

RefreshConfiguration re-gets the label configuration for the API to use.

Using Objective-C:

```
-(BOOL) refreshConfiguration:(NSError**)error;
```

Using either C or Objective-C:

```
BJClassifierRefreshConfiguration(BJClassifierAPIRef classifierAPI, CFErrorRef* error);
```

Parameters:

[in] classifierAPI

Type: BJClassifierAPIRef

Description: An API reference, as returned from BJClassifierAPICreate()

[out, optional] error

Type: error**

Description: If an error occurs, is populated with additional error information.

Return Value:

Type: bool

True if the configuration is refreshed, false if an error occurs.

If the call failed, the following may be returned in error:

- **Unlicenced**
- **Uninitialised**

4 SELECTOR OBJECT PROPERTIES

The Selector object (BJClassifierSelectorRef / id<BJClassifierSelectorProtocol>) contains the properties of a selector in a label configuration. The following section details the function calls to access these properties.

4.1 Name

The Name of the selector, in the current culture.

Objective-C:

```
@property (readonly, copy) NSString* name;
```

C / Objective-C

```
CFStringRef BJClassifierCopySelectorName(BJClassifierSelectorRef classifierSelector);
```

4.2 Identifier

the ID of the selector.

Objective-C:

```
@property (readonly, copy) NSString* identifier;
```

C / Objective-C

```
CFStringRef BJClassifierCopySelectorIdentifier (BJClassifierSelectorRef classifierSelector);
```

4.3 Selector Type

An enum representing the selector type:

Objective-C:

```
@property (readonly) enumBoldonJames_Classifier_eSelectorType type;
```

C / Objective-C

```
enumBoldonJames_Classifier_eSelectorType BJClassifierGetSelectorType(BJClassifierSelectorRef classifierSelector);
```

Return Value:

Type: enumBoldonJames_Classifier_eSelectorType

eSelectorTypes contain the following values:

- Invalid
- SingleSelect
- MultiSelect
- FreeText
- DateSelect
- DateOffset

4.4 IsValid

True if the selector is valid, false if not. A selector may be invalid if it is instantiated with an invalid Selector ID.

Objective-C:

```
@property (readonly) BOOL isValid;
```

C / Objective-C

```
bool BJClassifierGetIsSelectorValid(BJClassifierSelectorRef classifierSelector);
```

5 SELECTOR VALUE OBJECT METHODS

The Selector Value object (BJClassifierSelectorValueRef / id<BJClassifierSelectorValueProtocol>) contains the properties of a selector value within a selector. The following section details the function call to access these properties.

5.1 Name

The name of the selector value in the current culture.

Objective-C:

```
@property (readonly, copy) NSString* name;
```

C / Objective-C

```
CFStringRef BJClassifierCopySelectorValueName(BJClassifierSelectorValueRef classifierSelector);
```

5.2 Identifier

The ID of the selector value.

Objective-C:

```
@property (readonly, copy) NSString* identifier;
```

C / Objective-C

```
CFStringRef BJClassifierCopySelectorValueID(BJClassifierSelectorValueRef classifierSelector);
```

5.3 Colour

A string that represents the hex code for the selector value's colour.

Objective-C:

```
@property (readonly, copy) NSString* colour;
```

C / Objective-C

```
CFStringRef BJClassifierCopySelectorValueColour(BJClassifierSelectorValueRef classifierSelector);
```

5.4 Portion Marking

Returns the portion mark string if it exists. Otherwise, returns an empty string

Objective-C:

```
@property (readonly, copy) NSString* portionMark;
```

C / Objective-C

```
CFStringRef BJClassifierCopySelectorValuePortionMark(BJClassifierSelectorValueRef classifierSelector);
```

5.5 Alternate Name

The alternate name properties of the selector values (there are 3 separate alternate names). Each value is an empty string if no alternate name exists.

Objective-C:

```
@property (readonly, copy) NSString* altName1;

@property (readonly, copy) NSString* altName2;

@property (readonly, copy) NSString* altName3;
```

C / Objective-C

```
CFStringRef BJClassifierCopySelectorValueAltName1(BJClassifierSelectorValueRef classifierSelector);
```

```
CFStringRef BJClassifierCopySelectorValueAltName2(BJClassifierSelectorValueRef classifierSelector);

CFStringRef BJClassifierCopySelectorValueAltName3(BJClassifierSelectorValueRef classifierSelector);
```

5.6 Hierarchy Value

Determine whether or not the selector is hierarchal and if it is, where this value sits in the selector hierarchy. The property contains a numerical hierarch value for hierarchal selectors, and NULL for non-hierarchical selectors.

Objective-C:

```
@property (readonly, copy) NSNumber* hierarchyValue;
```

C / Objective-C

```
CFNumberRef BJClassifierCopySelectorValueHierarchy(BJClassifierSelectorValueRef classifierValue);
```

6 RETURN CODES

Most Classifier API functions will return a return code. The return code is an enum (**eReturnCode**) and specifies if the call succeeded and the reason for failure if it does not. Below is a list of all the return codes in the Classifier API:

Ok	Call was successful.
InvalidLicence	Licence provided to API is invalid.
Uninitialised	Function call failed because the API has not been initialised.
Unlicenced	Licence provided to the API is invalid or has expired.
InvalidConfig	API is not using a valid Classifier Configuration.
CannotDecodeLabel	The SISL passed in to the function cannot be read. Either it is not a valid SISL, or it belongs to a configuration not in use.
SelectorNotFound	The selector the API is looking for does not exist within a given SISL.
InvalidSelector	The selector is invalid i.e. null or empty.
InvalidSelectorValue	The selector value does not exist or is otherwise invalid.
InvalidFile	File does not exist or is otherwise invalid or corrupt.
InvalidHandle	The file handle is invalid.
CantWriteToFile	The file cannot be written to. It is either read-only or is currently being accessed by another process.
WrongSelectorType	Occurs when calling a multi-select function with a single-select selector specified as a parameter and vice versa.
BadCulture	Culture specified not recognised as a valid culture.
UnsupportedFileType	API does not support the reading or writing of this file type. Currently, the unsupported types are all Visio file types and can be enabled for read/write via ADS via the Classifier configuration.
CantWriteViaHandle	Unable to write to this file when passing the handle as a parameter. Try calling SetClassification() without a handle parameter.
InvalidCollection	Array passed as parameter is null, empty or otherwise invalid.
CannotCalculateHighWaterMark	Valid high-water mark label could not be determined from the collection of labels.
LabelsAreEqual	Labels are the same.
FirstLabelDominates	Label 1 > Label 2
SecondLabelDominates	Label 1 < Label 2
LabelsAreDifferent	Labels are not the same but neither is greater than the other.
Unknown	Unexpected exception thrown. Refer to BJTrace logging to help determine the cause.