Heurist API - v0.1

All classes objects in the Heurist API are accessible through the global namespace, and also the HAPI namespace. For example, HFooBar is also accessible as HAPI.FooBar; internally all names use the former naming scheme.

`class HRecord`

Instantiate class HRecord to create a Heurist record object. Personalisation data (e.g. ratings, personal tags) are included in these objects.

For details on using HRecord of record-type "relationship", see HRelationship.

Constructor

```
  HRecord()
```

Description

Creates a new, empty HRecord object.

Methods

```
  getID()  =>  string
```

If this HRecord corresponds to a record in the Heurist database, then return a unique identifier for that record; otherwise, null

```
  getVersion()  =>  int
```

The version of this record if it corresponds to an unmodified version of a record in the Heurist database; otherwise, null

```
  getTitle()  =>  string
```

Return the title of this record -- a string containing a useful combination of record details appropriate to the record type

```
  getURL()  =>  string
  setURL( string )  throws HPermissionException
```

The URL for this record if it exists; otherwise, null

```
  getNotes()  =>  string
  setNotes(string) throws HPermissionException
```

The notes (freeform data, from the scratchpad) for this record

```
  getRecordType()  =>  HRecordType
  setRecordType( HRecordType )  throws HInvalidRecordTypeException, HPermissionException
```

The type of this record (e.g. book, publisher, private notes). Note that a record of type "relationship" cannot be changed to another type, and vice versa -- an HInvalidRecordTypeException would be thrown.

```
getWorkgroup()  =>  HWorkgroup
setWorkgroup( HWorkgroup )  throws HInvalidWorkgroupException, HPermissionException
```

The workgroup to which this record belongs; otherwise, null. A record may only be assigned to a workgroup by a member of that workgroup. Alternatively, if the record belongs to a workgroup then it may only be removed from that workgroup by an administrator of that workgroup.

```
getNonWorkgroupVisible()  =>  boolean
setNonWorkgroupVisible( boolean )  throws HPermissionException
```

If this record belongs to a workgroup, then this determines whether or not the record is visible (but read-only) or invisible to Heurist users who are not in the workgroup.

```
getURLVerificationDate()  =>  Date
```

When was the record's URL last verified by Heurist's internal checking mechanisms?

```
getURLError()  =>  string
```

If the record's URL cannot be verified by Heurist, a string describing the error; otherwise, null

```
getCreationDate()  =>  Date

getModificationDate()  =>  Date
```

Respectively the creation and last-modified dates for this record

```
getHHash()  =>  string
```

A string -- the "Heurist hash" -- combining all the essential details used by Heurist to compare records. The HHash values for two records are the same if and only if Heurist would regard the records as duplicates.

```
getDetail( HDetailType )  =>  (value)  throws HValueException
```

If the given type is non-repeatable for this record type, returns its single value (or null if it is non-existent). If the value is repeatable, throws an HValueException.

```
getDetails( HDetailType )  =>  Array

addDetail( HDetailType, value )  throws HDetailVarietyMismatchException

removeDetails( HDetailType )
```

The record-type-specific details for this record. For a given HDetailType, there is a required detail variety (see HDetailType) -- if the supplied value to addDetail is not of the expected variety, then an HDetailVarietyMismatchException is thrown. "Non-truthy" values except 0 and false (e.g. null, undefined, NaN, "") will not be added.

```
setDetails( HDetailType, Array )  throws HDetailVarietyMismatchException
```

A convenience function equivalent to removeDetails(HDetailType) followed by repeated applications of addDetail(..)

```
changeDetail( HDetailType, oldValue, newValue )  throws
HDetailVarietyMismatchException, HValueException
```

If the value oldValue exists in this record for the given type, then change that value to newValue. Otherwise, an HValueException is raised.

```
isPersonalised()  =>  boolean  throws  HNotLoggedInException
```

Is this record in the logged-in user's favourites list?

```
addToPersonalised()  throws  HNotLoggedInException
removeFromPersonalised()  throws  HNotLoggedInException
```

Add/remove this record to/from the logged-in user's favourites list.

```
getPersonalNotes()  =>  string  throws  HNotLoggedInException, HNotPersonalisedException

setPersonalNotes(string)  throws  HNotLoggedInException, HNotPersonalisedException
```

A logged-in user's personal notes for this record. The user must have personalised the record (added it to their favourites), or else an HNotPersonalisedException is thrown.

```
getRating( HRatingType )  =>  HRatingValue  throws  HNotLoggedInException, HNotPersonalisedException

setRating( HRatingType, HRatingValue )  throws HNotLoggedInException, HNotPersonalisedException
```

A logged-in user's personal ratings for this record. The user must have personalised the record, or else an HNotPersonalisedException is thrown. The HRatingType and HRatingValue parameters are taken from the HRatings object.

```
getTags()  =>  Array[string]  throws  HNotLoggedInException, HNotPersonalisedException

addTag( tagString )  throws HUnknownTagException, HNotLoggedInException, HNotPersonalisedException

removeTag( tagString ) throws HNotLoggedInException, HNotPersonalisedException
```

A logged-in user's personal tags for this record. The user must have personalised the record, or else an HNotPersonalisedException is thrown. If an attempt is made to add a tag that is not already known as one of the user's existing tags, then an HUnknownTagException is thrown. Tags may be added and verified using the HTagManager class.

```
getKeywords()  =>  Array[HKeyword]
addKeyword( HKeyword )  throws HPermissionException
removeKeyword( HKeyword ) throws HPermissionException
```

Workgroup keywords for this record. HKeyword objects may only be instantiated by the HKeywordManager. Only keywords for workgroups of which the logged-in user is a member are visible.

```
getNotifications()  =>  Array[HNotification]  throws HNotLoggedInException, HNotPersonalisedException
```

Return an array of all the notifications associated with this record

```
addNotification( recipient, message, startDate, HFrequency )  => HNotification throws HNotLoggedInException, HNotPersonalisedException
```

Add a new notification to this record.

- `recipient` is an HUser instance, an HWorkgroup instance, or an email address (a string).

- `message` is a message to be included with the reminder mailout when it is sent

- `startDate` is a Date on which the reminder mail should be sent. If this value is null, the reminder is scheduled for immediate dispatch

- the final parameter is an HFrequency value indicating how often the reminder should be repeated after startDate

```
removeNotification( HNotification )  throws HNotLoggedInException
```
Remove the given notification from this record (has no effect if the notification is not associated with the record)

```
getComments()  =>  Array[HComment]  throws HNotLoggedInException
removeComment( HComment )  throws HPermissionException, HNotLoggedInException
```
Comments facilitate threaded discussion of a record. These methods deal only with top-level comments: see HComment for the similar functions allowing management of nested comments. See the HComment constructor for details on how to add a new comment to a record.

```
getRelatedRecords()  =>  Array[HRecord]
getRelationships()  =>  Array[HRelationship]
```
Relationships between records in Heurist are represented by another record, allowing the addition of complex metadata. If only the records related to this current record are required, then getRelatedRecords() is used. If the relationship records are required also (e.g. to allow examination of relationship type) then getRelationships() is used. **Only records that are locally cached in the same HStorageManager as this record are known to getRelatedRecords() / getRelationships().**

```
isReadOnly()  =>  boolean
```
Returns true if this record's public details cannot be modified by the currently logged-in user.

```
isModified()  => boolean
```
Return true if the given record has been modified since it was loaded through its storage manager. Note that this will always return true if the record was not loaded through a storage manager (i.e. it has been created afresh).

```
isValid()  =>  boolean
```
Returns true if this record has all required details.

```
lock()
unlock()
isLocked()  =>  boolean
```
Ordinarily, if a record is already cached by an HStorageManager, then reloading it will overwrite any local changes. This default action is prevented by locking the record.

class HRelationship extends HRecord

Heurist supports detailed relationships between records, by providing a "relationship" record-type. HAPI provides extra functionality for records of this type. Note that relationship records must be created with the HRelationship constructor; they cannot be created using the HRecord interface.

Constructor
```
  HRelationship( primaryRecord, relationshipType, secondaryRecord )  throws
HBadRelationshipException
```
Description

Creates a new relationship between the two records, with type "relationshipType". Valid relationship-types are available from the static method HRelationship.getRelationshipTypes()

Methods
```
getPrimaryRecord()  =>  HRecord
```

Return the "primary" record associated with this relationship

`getSecondaryRecord()  =>  HRecord`

Return the "secondary" (non-primary) record associated with this relationship

`getType()  =>  string`

Return the type of relationship of the primary record to the secondary record, e.g. primary "isParentOf" secondary

`getInverseType()  =>  string`

Return the type of relationship of the secondary record to the primary record, e.g. secondary "isChildOf" primary

Static methods

`getRelationshipTypes()  =>  Array[string]`

Return an array of all valid relationship types

class HNotes extends HRecord

Heurist allows the creation of objects known as "private notes". These objects are similar to records, but cannot have details, workgroup keywords, notifications, comments or relationships attached. They are only visible to the currently logged-in user. The HNotes class overloads getNotes() / setNotes(..) to call getPersonalNotes() / setPersonalNotes(..) respectively. Calls to the functions for managing details, keywords etc will raise an HInvalidRecordTypeException.

object HTagManager

The HTagManager keeps track of the logged-in user's personal tags attached to records. Each tag contains a string from a semi-restricted vocabulary. These tags are specific to each user, but are visible to all users in search queries.

Methods

`getAllTags()  =>  Array[string]  throws HNotLoggedInException`

Returns an array of all this user's tags.

`getTag( string )  =>  string  throws HNotLoggedInException`

If the given string matches one of this user's existing tags, then return the existing tag; otherwise, null. Strings are considered to match if they are equal when expressed in canonical form -- this canonical form ignores case, and squashes whitespace and punctuation.

`getMatchingTags( string [, count] )  =>  Array[string]  throws HNotLoggedInException`

Return a list of tags which are similar to the given string. If count is supplied, then exactly that number of tags are returned (subject to the entire size of the vocabulary); otherwise an appropriate number of nearby matches are returned, perhaps none if there are no suitable tags.

`addTag( string )  throws HNotLoggedInException`

Adds the given string to this user's tag vocabulary, if it is not already there. Note that such changes are restricted to the local HTagManager -- if there are multiple instances of the HTagManager / HAPI running simultaneously on the same machine (or different machines) then tags added to one instance will not be available to others.

object HKeywordManager

The HKeywordManager is a convenience used to manage the keywords for workgroups of which the logged-in user is a member.

Methods

`getAllKeywords()  =>  Array[HKeyword]  throws HNotLoggedInException`

Return an array of all the keywords from all workgroups of which this user is a member.

`getWorkgroupKeywords( HWorkgroup )  =>  Array[HKeyword]  throws`

`HNotLoggedInException, HInvalidWorkgroupException`

Return an array of all the keywords from the provided workgroup (of which this user must be a member).

class HKeyword

HKeyword instances correspond to workgroup keywords. As these are controlled by workgroup administrators, they cannot be constructed via the HAPI, and are only accessible through the HKeywordManager instance.

Methods

`getName()  =>  string`

Return the name of this keyword

`getWorkgroup()  =>  HWorkgroup`

Return the workgroup to which this keyword belongs

object HWorkgroupManager

The HWorkgroupManager is a convenience used to manage the HWorkgroup objects corresponding to the Heurist workgroups.

Methods

`getUserWorkgroups()  =>  Array[HWorkgroup]`

Return all the workgroups of which the logged-in user is a member.

class HWorkgroup

An HWorkgroup instance corresponds to a Heurist workgroup. These cannot be constructed via the HAPI, and are only accessible through the HWorkgroupManager instance.

Methods

`getName()  =>  string`

Return a short name for this workgroup

`getLongName()  =>  string`

Return a longer name for this workgroup

`getDescription()  =>  string`

Return a descriptive tract about this workgroup

`getURL()  =>  string`

Return a URL for this workgroup

object HColleagueGroupManager

The HColleagueGroupManager is a convenience used to manage the HColleagueGroup objects corresponding to Heurist colleague-groups.

Methods

`getColleagueGroups()  =>  Array[HColleagueGroup]`

Return all the logged-in user's colleague groups.

class HColleagueGroup

An HColleagueGroup instance corresponds to a Heurist colleague-group. These differ from workgroups inasmuch as they are defined by the user (although this is not currently available through the HAPI).

Methods

getName() => string

Return the name for this colleague-group

object HRatings

The HRatings object contains constants for manipulating personalised record ratings.

Constants

```
CONTENT
INTEREST
QUALITY
```

The (orthogonal?) types of rating available to users for each record

```
CONTENT_1
CONTENT_2
CONTENT_3
CONTENT_4
CONTENT_5
```

Human-readable strings corresponding to the different content ratings; CONTENT_1 is a low rating, CONTENT_5 is a high rating

```
INTEREST_1
INTEREST_2
INTEREST_3
INTEREST_4
INTEREST_5
```

Human-readable strings corresponding to the different interest ratings; INTEREST_1 is a low rating, INTEREST_5 is a high rating

```
QUALITY_1
QUALITY_2
QUALITY_3
QUALITY_4
QUALITY_5
```

Human-readable strings corresponding to the different quality ratings; QUALITY_1 is a low rating, QUALITY_5 is a high rating

object HRecordTypeManager

The HRecordTypeManager is a convenience for dealing with Heurist record types.

Methods

`getRecordTypes()  =>  Array[HRecordType]`

Return an array of all the available record types

`getRecordTypeById( integer )  =>  HRecordType`

Return the record type associated with the given id (as used in Heurist searches etc); if no type is associated with that id, then null. This is the recommended method for obtaining a previously known record type object, as record names are subject to change.

`getRecordTypeByName( string )  =>  HRecordType`

Return the record type with the given name; if there is not exactly one type associated with that name, then null.

class HRecordType

HRecordType instances correspond to Heurist record types. As these are controlled by Heurist administrators, they cannot be constructed via the HAPI, and are only accessible through the HRecordTypeManager instance.

Methods

`getID()  =>  integer`

Return the unique ID associated with this record type, as used in Heurist searches

`getName()  =>  string`

Return a string describing this record type

object HDetailManager

The HDetailManager is a convenience for dealing with Heurist record details, detail types, and associated concepts.

Methods

`getDetailTypesForRecordType( HRecordType )  =>  Array[HDetailType]`

Return an array of all the record detail types specific to the given record type. The returned types are either *required*, *recommended* or *optional* (but never those explicitly *excluded*). Their order is as they are presented on the Heurist edit form.

`getRequiredDetailTypesForRecordType( HRecordType )  =>  Array[HDetailType]`
`getRecommendedDetailTypesForRecordType( HRecordType )  =>  Array[HDetailType]`
`getOptionalDetailTypesForRecordType( HRecordType )  =>  Array[HDetailType]`

Return an array of all the record detail types required / recommended / optional for the given record type.

`getDetailRequirement( HRecordType, HDetailType )  =>  HRequirement`

Return the requirement (e.g. required, optional) of the given detail type with the given record type

`getDetailRepeatable( HRecordType, HDetailType )  =>  boolean`

Can the given detail type hold more than one value for the given record type?

`getDetailNameForRecordType( HRecordType, HDetailType)  =>  string`
`getDetailPromptForRecordType( HRecordType, HDetailType)  =>  string`

Some detail types have more descriptive names / prompts depending on which record type they are used with. These methods return the record-type-specific value if available, or HDetailType::getName() / ::getPrompt() otherwise

class HDetailType

Bibliographic records in Heurist may contain record-type-specific details; each detail is of a specific type, represented in HAPI by the HDetailType class. New HDetailType objects cannot be instantiated, but are accessible through the HDetailManager.

Methods

`getName()  =>  string`

Return a short name used to describe this detail type

`getPrompt()  =>  string`

Return a short string which could be useful to users creating a new value of this type `getVariety()  =>  HVariety`

Return the expected variety (e.g. literal, geographic value) for this type

`checkValue( value )  =>  boolean`

Return true if the given value is appropriate for this type

`getEnumerationValues()  =>  Array[string]`

If this type is of an enumeration variety, return an array of strings which are valid values; otherwise, null

`getConstrainedRecordType()  =>  HRecordType`

If this type is of a record reference variety, and constrained to a particular record type, then return that record type; otherwise, null

object HVariety

HVariety is a collection of constants representing the different kinds of detail type.

Constants

`LITERAL`

Corresponds to details with a simple value such as a string, integer, float or boolean

`DATE`

Corresponds to details with a date value

`ENUMERATION`

Corresponds to details taking a value from some type-specific vocabulary (see HDetailType::getEnumerationValues())

`REFERENCE`

Corresponds to details taking a reference to a Heurist record

`FILE`

Corresponds to details referencing a file uploaded to the Heurist repository

`GEOGRAPHIC`

Corresponds to details with a geographic value (see HGeographicValue)

object HRequiremence

HRequiremence is a collection of constants representing how essential a particular detail type is to a particular record type. Also, requiremence is not a real word.

Constants

`REQUIRED`

`RECOMMENDED`

`OPTIONAL`

`FORBIDDEN`

The different requiremences known to Heurist.

class HNotification

An HNotification object represents a time-delay reminder message about a record, sent via email from the Heurist system. There is no explicit constructor available through HAPI; rather use the factory method HRecord::addNotification(...)

Methods

`getRecord()  =>  HRecord`

Return the record with which this notification is associated

`getWorkgroupRecipient()  =>  HWorkgroup`

getColleagueGroupRecipient() => HColleagueGroup

`getUserRecipient()  =>  HUser`

`getEmailRecipient()  =>  string`

Return the recipient of this reminder. Exactly one of these will return a non-null value.

`getMessage()  =>  string`
Return the text message associated with this notification
`getStartDate()  =>  Date`
Return the first date on which the notification has been / will be sent
`getFrequency()  =>  HFrequency`
Return the frequency with which notifications will be repeated after the start date

object HFrequency

HFrequency is a collection of constants representing frequencies at which one may request a notification being re-sent. The values of the constants are appropriately human-readable.

Constants

```
ONCE
DAILY
WEEKLY
MONTHLY
ANNUALLY
```

Static methods

`isValidFrequency( value )  =>  boolean`
Return true if the given value is a known HFrequency value

class HComment

An HComment represents one portion of a threaded discussion pertaining to a Heurist record.

Constructors

`HComment( HRecord )  throws HNotLoggedInException, HPermissionException`
`HComment( HComment )  throws HNotLoggedInException, HPermissionException`
Creates a new, empty comment. If the argument to the constructor is an HRecord, then the new object becomes a top-level comment attached to that record. If the argument is an HComment, then the new object becomes a reply to the specified comment, attached to the same record as the specified comment. An HNotLoggedInException is thrown if there is no currently logged-in user; an HPermissionException is thrown if there is a user-based restriction (e.g. workgroup-restricted) on the record to which the comment is being attached, which the currently logged-in user does not satisfy.

Methods

`getText()  =>  string`
`setText( string )  throws HPermissionException`
Return / set the content of the comment. An HPermissionException is thrown if an attempt is made to set the text of a comment that was not created by the currently logged-in user.

`getParent()  =>  HComment`
Return the parent comment if this is a reply; if this is a top-level comment then return null
`getReplies()  =>  Array[HComment]`
Return an array of the comments that are a direct reply to this comment.
removeReply(HComment) throws HCommentMismatchException
Remove the given reply from this comment; an HCommentMismatchException is thrown if the supplied HComment is not a direct reply to this comment.

`getCreationDate()  =>  Date`
`getModificationDate()  =>  Date`
Return the creation / modification dates of this comment as stored in the Heurist database. If the comment has not been saved to the database, then both return null; if the comment has not been modified from its original state in the database, then getModificationDate() returns null.

`getUser()  =>  HUser`
Return the user who created this comment
`getRecord()  =>  HRecord`

Return the record with which this comment is associated.

class HException

All exceptions in HAPI are instances of subclasses of HException.

class HPermissionException

class HInvalidWorkgroupException

class HInvalidRecordTypeException

class HDetailVarietyMismatchException

class HBadRelationshipException

class HNotLoggedInException

class HNotPersonalisedException

class HUnknownTagException

interface HStorageManager
object HeuristScholarDB

An HStorageManager is a cache for Heurist objects, with connectivity to a persistent record storage facility.
There is one pre-defined implementation of HStorageManager available as `HeuristScholarDB` ; this connects to the central Heurist database at heuristscholar.org.
Methods

`getRecord( id )  =>  HRecord`

If the record with the given Heurist ID (as returned by HRecord::getID()) is present in the local cache, return it; otherwise, return null

`saveRecord( HRecord [, HSaver] )`

Commence an asynchronous write of the given record to persistent storage. If an HSaver is supplied, it will contain callbacks for success and error handlers; otherwise, a default HSaver is used. Upon the success of a saveRecord, the HRecord's version (and possibly record ID) will be updated to match the persistent storage. The HRecord cannot be changed while the save operation is progress.

`loadRecords( HSearch [, HLoader] )`

Commence an asynchronous read of the record(s) matching the given HSearch from persistent storage into the storage manager. If an HLoader is supplied, it will contain callbacks for success and error handlers; otherwise, a default HLoader is used. If a record has been loaded already, any local modifications will be overwritten unless the record is locked (see HRecord::lock / HRecord::unlock).

`removeRelationship( HRelationship [, HSaver] )  throws HPermissionException`

Remove the given relationship from Heurist. Since this removal is saved to persistent storage, an HSaver object may be supplied to allow notification of the success or failure of the removal. Update of the storage-manager's cache will not occur until successful save is notified (this happens even if no HSaver is specified). After update of the local cache, the removed HRelationship object is invalid, and any subsequent attempts to use methods in the removed HRelationship will cause an HBadRelationshipException to be thrown.

`saveFile( FileInputElement [, HSaver] )`

Upload the user-selected file in the given HTML <FILE> element to the Heurist system. If an HSaver is supplied, it will contain callbacks for success and error handlers.

`findFiles( HFileSearchOptions, HLoader )`

Commences an asynchronous search for files in the Heurist system matching the given specification.

class HSaver

An HSaver is a lightweight wrapper associated with a single save of a Heurist object (e.g. HRecord, HFile) by an HStorageManager. See also HLoader.

Constructor

`HSaver( onsaveCallback [, onerrorCallback] )`

Description

A new, reusable, HSaver is initialised with the given handlers. In general, the HSaver is passed to a save method in an HStorageManager, along with a Heurist object to be saved.

- Upon a successful save, onsaveCallback will be invoked with the saved object as its single argument
- Otherwise, onerrorCallback will be invoked with two arguments: the unsaved object, and a string describing the failure

Both callbacks are invoked with the HSaver object as *this*.
See (for example) HStorageManager::saveRecord(..)

class HLoader

An HLoader is a lightweight wrapper associated with the loading of Heurist objects from an HStorageManager. See also HSaver.

Constructor

`HLoader( onloadCallback [, onerrorCallback] )`

Description

Creates a new, reusable, HLoader is initialised with the given handlers. In general, the HLoader is passed to a load method in an HStorageManager, along with a descriptor object specifying the objects to be loaded.

- Upon the successful asynchronous loading of object(s), onloadCallback will be invoked two arguments: the descriptor object, and an array of loaded objects
- otherwise, onerrorCallback will be invoked with two arguments: the descriptor object, and a string describing the failure

Both callbacks are invoked with the HLoader object as *this*.
See (for example) HStorageManager::loadRecords(..)

class HSearch

An HSearch object is associated with a search of Heurist records.

Constructor

```
  HSearch( query [, options] )
```

Description
Creates a new search object based on the given query string, with options as specified in HSearchOptions. The query string is as supplied to the main Heurist search page. The resulting HSearch object may then be used in a call to HStorageManager::loadRecords(..) to return a subset of the records managed by that HStorageManager.

class HSearchOptions

Instances of this class are used in the options argument to the constructor of the HSearch class. There is no constructor for this class. Instead, this class is instantiated as a javascript object literal. All options are, uh, optional.

Properties

```
  favourites-only: boolean
```

If true, only records that have been personalised by the logged-in user are returned. (default: false)

```
  recent-only: boolean
```

If true, only records that have been accessed recently by the logged-in user are returned. (default: false)

object HWikiManager

Convenience to manage the wikis associated with a Heurist record.

Methods

```
  getWikis( HRecord )  =>  Object
```

Return an Object containing property => value pairs, where every property is the name of a wiki associated with the supplied record, and the corresponding value is the URL of that wiki.
// possibly in future: getWikiSummary(wikiName) => string

object HUserManager

The HUserManager is a convenience for searching registered users of the Heurist system.

Methods

```
  findUser( name )  =>  Array[ HUser ]
```

Return an array of users matching the given name. Some degree of fuzziness is used in the search; the size of the result-set is internally limited.

class HUser

Instances of this class represent a registered user of the Heurist system. Necessarily there is little publicly-exposed information through HAPI. HUser objects cannot be instantiated by HAPI, but users can be searched using the HUserManager.

Methods

```
  getUsername()  =>  string
```

Return the login-name of this user

```
  getRealName()  =>  string
```

Return the OMG IRL name of this user

object HCurrentUser

HCurrentUser is an HUser instance corresponding to the currently logged-in user, with some more specific information about capabilities and preferences. If the user is not logged in, then calls to any methods except isLoggedIn() throw an HNotLoggedInException.

Methods

`getUsername()  =>  string  throws HNotLoggedInException`

Return the login-name of the logged-in user

`getRealName()  =>  string  throws HNotLoggedInException`

Return the "real life" name of the logged-in user

`isLoggedIn()  =>  boolean`

Whether or not there is a "current user"

`isAdministrator()  =>  boolean throws HNotLoggedInException`

Whether or not the current user is an administrator

`getDisplayPreference( name )  =>  string throws HNotLoggedInException`

Return the current user's setting for the given named preference (e.g. "show-help"). If the named preference is not known to Heurist, return null.

class HGeographicValue

An HGeographicValue object represents a bibliographic detail of the geographic variety. HGeographicValue is not really useful for creating or manipulating such objects: any new geographic objects should be constructed using the Geographic Object Interface (GOI) and its geographic-type-specific objects.

Constructor

HGeographicValue(HGeographicType, wkt-string)

Create a new geographic value of the given type, using the given raw geographic data

Methods

`getType()  =>  HGeographicType`

Return the type of geographic value this object represents

`getWKT()  =>  string`

Return the raw geographic data for this object in OpenGIS "Well Known Text" format

`toString()  =>  string`

Return a compact representation of this object, incorporating its type and bounds

object HGeographicType

HGeographicType is a collection of constants representing the different kinds of geographic value.

Constants

`BOUNDS`

A rectangular region (minimum / maximum longitude and latitude)

`CIRCLE`

A nominally "circular" region (a central point with a radius assuming distance-equivalence of latitude and longitude)

`POLYGON`

A contiguous region defined by at least three vertices

`PATH`

A series of connected points with an associated directionality

`POINT`

A single latitude + longitude

class HFile

An HFile object represents a file that has been uploaded to the Heurist system, and can be attached to Heurist records as a bibliographic detail of the file variety. HFile has no constructor: previously uploaded files may be exposed as HFile objects using HStorageManager::findFile, or new files may be uploaded using HStorageManager::saveFile.

Methods

`getOriginalName()  =>  string`

Return the name of the uploaded file originally uploaded to the Heurist system, as reported by the uploader's computer

`getSize()  =>  int`

Return the size of the file in octets (bytes)

`getType()  =>  string`

Return a "best guess" at the MIME type of the file, e.g. "image/jpeg"

`getURL()  => string`

Return a URL from which the file may be downloaded

`getThumbnailURL()  =>  string`

Return a URL for a thumbnail image of the file

class HFileSearchOptions

Instances of this class are used to specify searches for file objects in an HStorageManager. There is no constructor for this class. Instead, this class is instantiated as a javascript object literal. At least one option must be specified.

Properties

`file-name: string`

A pattern to be used in matching against the name of the file as it was uploaded to the Heurist system. Normal UNIX glob-style semantics apply.

`file-type: string`

A MIME-type to restrict the matching files. It is of the usual form "type/sub-type". An asterisk (*) may be used as a wildcard in place of sub-type.